Joni Purojärvi

Lääkinnällisen laitteen ohjelmistokehitys täydennetyllä Scrum-mallilla

Tietotekniikan
(Ohjelmistotekniikka)
pro gradu -tutkielma
18. lokakuuta 2010

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Joni Purojärvi
Yhteystiedot: joni.purojarvi@jyu.fi
Työn nimi: Lääkinnällisen laitteen ohjelmistokehitys täydennetyllä Scrum-mallilla
Title in English: Medical device software development with enhanced Scrum
Työ: Tietotekniikan (Ohjelmistotekniikka) pro gradu -tutkielma
Sivumäärä: 106
Tiivistelmä: Tässä tutkielmassa perehdytään Scrum-mallin käyttöön lääkinnällisen laitteen ohjelmistokehityksessä. Lääkinnällisissä laitteissa olevien ohjelmistojen määrä on kasvanut viime vuosina paljon ja potilasvahinkojen myötä viranomaiset ovat asettaneet vaatimuksia lääkinnällisen laitteen ohjelmistokehitykselle. Tutkimuksessa tunnistetaan Euroopan unionin lääkinnällisten laitteiden direktiivin asettamat vaatimukset ohjelmistokehitykselle. Tärkeimmät vaatimukset liittyvät turvalliseen suunnitteluun, jota ohjataan riskienhallinnalla ja varmistetaan testauksella. Scrum-mallia täydennetään tunnistettujen vaatimusten pohjalta ja lopuksi esitetyn mallin arvioidaan täyttävän sille asetetut vaatimukset ja noudattavan ketterien menetelmien arvoja.
English abstract: This thesis focuses on medical device software development with Scrum model. The volume of software as an overall part of medical devices has been increasing over the last few years and due to mishaps in treatments, the authorities have set requirements for medical device software development. This study identifies requirements set to software development by the European Union medical devices directive. Medical devices are required to be designed in such way that they are safe and effective. The safe design is controlled by risk management and verified with testing. Scrum model is then supplemented with the identified requirements. The presented model is assessed against the identified requirements and values stated in the Agile manifesto.
Avainsanat: lääkinnällinen laite, Scrum, ohjelmistokehitys, riskienhallinta, verifiointi, validointi, IEC 60601, IEC 62304, ISO 14971, ISO 13485
Keywords: medical device, Scrum, software development, risk management, verification, validation, IEC 60601, IEC 62304, ISO 14971, ISO 13485

Esipuhe

Kiinnostus tämän tutkielman aihepiiriin syntyi työn ohessa Sysdrone Oy:llä. Sain olla mukana kehittämässä Scrum-mallia Sysdronen käyttöön haastavissa ohjelmistoprojekteissa ja samalla sain mahdollisuuden tutustua lääkinnällisten laitteiden ohjelmistokehitykseen. Erityiset kiitokset haluan esittää Ilkka Laitiselle, Jani Lehtiselle ja Matti Lehtiselle, jotka ovat antaneet oman näkökulmansa tässä tutkielmassa esitettyyn Scrum-malliin. Lisäksi haluan kiittää Johanna Hakkaraista tutkielman kirjoitusasun tarkastamisesta sekä ohjaajiani Jonne Itkosta ja Tommi Kärkkäistä uusista näkökulmista aiheeseen.

Lääkinnällisten laitteiden direktiivin ymmärtäminen ja eri standardeihin tutustuminen vaativat suurimman osan tähän tutkielmaan käytetystä ajasta. Lakiteksti ja standardit ovat melko puisevaa luettavaa ja pikaisesti ajateltuna hyvin kaukana käytännön tekemisestä. Tutkielman kirjoittamisen aikana minulle kuitenkin valkeni, että standardit on kirjoitettu opastamaan tekemistä. Kaikkea ei tarvitse keksiä itse, joten miksi ei käyttäisi tietoa, joka on kirjoitettu standardeihin. Näiden standardien sovittaminen omaan ohjelmistokehitysmalliin voi olla tosin haasteellista – varsinkin kun valitettavan useissa standardeissa viitataan ohjelmistokehityksen malliin edelleen vesiputousmallina. Onneksi tästä vesiputousmallin ideasta ollaan hiljalleen luopumassa, kuten standardissa IEC 62304, jossa kehotetaan käyttämään iteratiivista mallia.

Tutkielmassa esitetään yksi näkökulma kuinka verifiointi, riskienhallinta ja ”raskas” jäljitettävyys voidaan toteuttaa Scrum-mallissa. En tehnyt mallista liian tarkkaa kuvausta, koska haluan että malli ei rajoita tiettyihin työkaluihin tai mikrotason käytäntöihin. Toivon, että tästä työstä on apua myös muille, jotka työskentelevät lääkinnällisten laitteiden ohjelmistokehityksen parissa ja haluavat parantaa omaa ohjelmistokehitysmalliaan.

Sanasto

IEEE Institute of Electrical and Electronics Engineers. Kansainvälinen järjestö, joka julkaisee monia tekniikan alan tiedejulkaisuja ja määrittelee standardeja.

IEC International Electrotechnical Commission. Kansainvälinen sähköalan standardointiorganisaatio.

Ilmoitettu laitos Jonkin Euroopan maan viranomaisen nimittämä puolueeton laitos, jolle on myönnetty lupa suorittaa ilmoitetun laitoksen tehtäviä, kuten vaatimuksenmukaisuusarviointeja.

ISO International Organization for Standardization. Kansainvälinen standardisoimisjärjestö.

Kliininen tutkimus Lääkkeen tai muun hoitomuodon tehon ja turvallisuuden osoittamiseen tähtäävä tutkimus.

Lääkinnällinen laite Laite tai ohjelmisto, jota käytetään ihmisten sairauden diagnosointiin, ehkäisyyn, tarkkailuun, hoitoon tai lievitykseen.

Riski Vahingon todennäköisyyden ja vakavuuden yhdistelmä.

Riskienhallinta Projektin tai ohjelmistosta aiheutuvien riskien järjestelmällinen määrittely ja niihin varautuminen.

Vahinko Fyysinen vamma, terveyshaitta tai omaisuusvahinko.

Vakavuus Toteutuneen vaaran seuraamusten mittaus.

Validointi Prosessi, jossa varmistetaan, että ohjelmiston määrittely vastaa odotuksia ja vaatimuksia.

Verifiointi Prosessi, jossa varmistetaan, että ohjelmisto vastaa määrittelyä.

Vähimmäistämistoimenpide Toimenpide, jolla riski estetään tai vahingon todennäköisyyttä pienennetään.

Sisältö

1 Johdanto

Lääkintälaitteissa olevien ohjelmistojen määrä ja laajuus ovat kasvaneet viime vuosina paljon [10, s. 3]. Samoin ohjelmistoista on tullut turvallisuuskriittisiä muuttujia lääkinnällisen laitteen käytössä. Tunnetuin tapaus ohjelmistovirheen aiheuttamista potilasvahingoista lienee Therac-25-sädehoitolaitteen tapaturmat. Therac-25:n ohjelmistossa oleva vika aiheutti liian suuren säteilyannoksen hoitojakson aikana, jonka seurauksena viisi ihmistä kuoli ja kuusi sai elinikäisiä vammoja. Onnettomuuksien tutkinnassa viaksi tunnistettiin ohjelmointivirhe, joka olisi voitu estää turvallisella suunnittelulla ja kattavammalla testauksella [85, s. 1-50]. Toinen tunnettu tapahtuma on erään potilastietokantajärjestelmän käyttöliittymän suunnitteluvirhe, jonka seurauksena sekoitettiin kaksi samannimistä potilasta. Virheen vuoksi potilas sai vääriä lääkkeitä ja menehtyi myöhemmin [23].

Ohjelmistojen roolin merkityksen kasvun ja potilasvahinkojen seurauksena ohjelmistojen turvallisuuteen kiinnitetään enemmän huomiota. Laitteet on suunniteltava ja valmistettava siten, että ne eivät vaaranna potilaan turvallisuutta suunnitellussa käyttötarkoituksessa. Ohjelmistojen arviointi tapahtuu tiettyjen turvallisuus- ja riskipohjaisten standardien mukaan. Standardit asettavat vaatimuksia suunnitteluprosessille, joten turvallisuustekijät on otettava huomioon jo tuotteen kehityksen elinkaaren alussa [16, s. 3-4].

Ohjelmistojen laajuus ja muuttuvat vaatimukset muodostavat suuria riskejä perinteisissä ohjelmistokehitysprosesseissa. Vaatimusmäärittely on harvoin täydellinen sovelluskehityksen alussa ja valmiin ohjelmiston muuttaminen on paljon kalliimpaa kuin muutoksiin varautuminen jo heti kehityksen alussa [65, s. 76]. Ketterät menetelmät ratkaisevat osan perinteisistä ohjelmistokehityksen ongelmista. Lakisääteiset vaatimukset aiheuttavat kuitenkin omat haasteensa ketterien menetelmien käyttöön lääkintälaitteiden kehityksessä. Tunnetut ketterät menetelmät, kuten XP ja Scrum, eivät ota kantaa riskienhallintaan tai verifiointiin ja validointiin ohjelmistokehityksen aikaisina tehtävinä.

Tämän työn tutkimustavoitteena on osoittaa kuinka Scrum-mallia voidaan käyttää lääkinnällisen laitteen ohjelmistokehityksessä. Tätä tarkoitusta varten Scrum-mallia täydennetään tarkennetulla analyysivaiheella, esitellään erillisen riskinhallintaprosessin integraatio malliin sekä määritellään tarkemmin verifiointi- ja validointitoimenpiteet kehityksen eri vaiheissa. Samoin esitetään kuinka ja missä vai-

heessa eri toimenpiteet ja niiden tulokset dokumentoidaan. Lopuksi esitetty malli arvioidaan: onko malli vielä Scrum, täyttyvätkö Euroopan unionin direktiivin asettamat säännökset ohjelmistokehitykselle ja mitä haasteita mallissa on.

Luvussa 2 kuvataan ohjelmistokehityksen tehtäviä ja luvussa 3 esitetään Scrum-ohjelmistokehitysmalli. Vaatimusmäärittelyn merkitystä ohjelmistokehityksessä ja sen haasteita esitetään luvussa 4. Lainsäädännön asettamia vaatimuksia lääkinnällisen laitteen ohjelmistokehitykselle kuvataan luvussa 5. Scrum-mallia täydennetään lakisääteisten vaatimuksien pohjalta. Täydennetty malli on esitelty muodollisesti luvussa 6 ja seikkaperäisesti esimerkin avulla luvussa 7. Lopuksi täydennettyä Scrum-mallia arvioidaan luvussa 8 lakisääteisten vaatimusten, Scrum-mallin ja ketterien menetelmien näkökulmasta.

2 Ohjelmistokehitys

Tässä kappaleessa esitellään ohjelmistokehityksen piirteitä ja kuvataan vaiheita V-mallin sekä spiraalimallin näkökulmista. V-malli on valittu esiteltäväksi tähän tutkielmaan, koska se on perinteinen suunnittelu- ja dokumenttivetoinen ohjelmistokehitysmalli, jossa ohjelmistoa koskevat vaatimukset kehitetään ja dokumentoidaan hyvin ennen toteutusta. Useat standardit, joita esitellään myöhemmin tässä tutkielmassa, viittaavat ohjelmistokehitysmalleihin tällaisina. Spiraalimalli esitellään tutkielmassa, koska se on iteratiivinen ohjelmistokehitysmalli, kuten myös myöhemmin esiteltävä Scrum. Spiraalimalli ottaa myös kantaa riskienhallintaan, joka on tärkeä osa lääkinnällisen laitteen ohjelmistokehitystä.

2.1 Ohjelmistokehityksestä ja malleista

Ohjelmistokehitysmallien juuret juontavat aina 1960-luvun lopulle silloisen ohjelmistokriisin aikaan. Tällöin ohjelmistojen koko kasvoi samoin kuin tietokoneiden tehot moninkertaistuivat. Ohjelmistokehityksessä ei muista insinöörialoista poiketen ollut käytössä yleisesti tunnettuja, kehittyneitä ja määriteltyjä malleja [66, s. 5-21]. Ensimmäiset ohjelmistokehitysmallit syntyivät 1970-luvulla [64] ja samaan aikaan esiteltiin käsite ohjelmistotuotanto.

Ohjelmistotuotannolla tarkoitetaan kokonaisvaltaista käsitystä ohjelmistojen kehittämisestä. Käsitteen alle kuuluu myös varsinaisen ohjelmistokehityksen ulkopuolisia prosesseja, kuten laatujärjestelmä, tuotteenhallinta, projektinhallinta ja laadunvarmistus. IEEE [35] määrittelee ohjelmistotuotannon suomennettuna seuraavasti:

1. Systemaattinen, kurinalainen ja mitattava sovellutus ohjelmiston kehitykseen, käyttöön tai ylläpitoon.

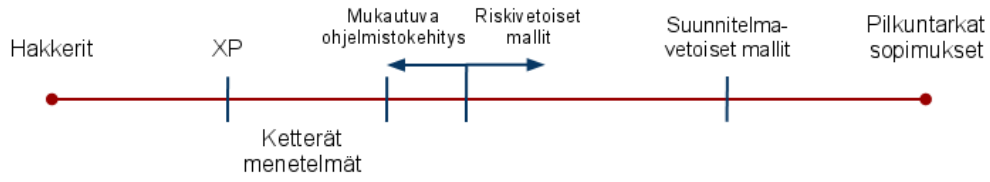
2. Edellä esitetyn sovellutusten tutkiminen.

Ohjelmistokehitysmallit kuvaavat ohjelmiston kehitykseen tarvittavat toimet. Ohjelmistokehitystä tehdään määritellyissä vaiheissa ja tämä ajatus on peräisin jo 1950-luvulta [54]. Ohjelmistokehitykselle voidaan nähdä muutamia ominaisia vaiheita, joita ovat Sommervillen mukaan [58, s. 12] suunnittelu, kehitys, validoin-

ti ja evoluutio. Evoluutiolla hän tarkoittaa ohjelmiston muuntautumista asiakasvaatimuksia vastaavaksi. Winston Royce kuvaa ohjelmistokehityksen vaiheita, niiden riippuvuuksia ja riskejä 1970-luvulla julkaisemassaan artikkelissa [56]. Paperissa esitellään yhdessä kuvassa kehityksen vaiheet perättäisinä askeleina ja tästä on ajan saatossa syntynyt käsite vesiputousmallista. Todellisuudessa Roycen artikkelissa kuvataan iteratiivista kehitystyötä sen ajan kontekstissa [64, s. 48].

Humphrey ja Kellner esittävät, että prosessimallit ovat projektikohtaisia, mutta ne tarvitsevat tuekseen korkeamman tason kehyksen mallista [57, s. 2]. Kehyksellä määritellään ohjelmistokehityksen vaiheet ja toimet, jotka tukevat kehitystä. Malleja muokataan vain jos kyseinen projekti erityisesti sitä vaatii. He esittävät, että näiden mallien tulee: kuvata kuinka työ oikeasti tehdään, olla esitettynä ymmärrettävässä muodossa ja olla muokattavissa millä tahansa tarkkuustasolla [57, s. 3].

Ohjelmistokehitysmallit voidaan jakaa perinteisiin malleihin ja ketteriin menetelmiin. Perinteisiä malleja on muun muassa spiraali-, protoilu- ja V-malli. Ketteriä menetelmiä ovat puolestaan XP (eXtreme Programming), Scrum ja DSDM (Dynamic System Development Model). Yksiselitteistä eroa perinteisten mallien ja ketterien menetelmien välille on hankala tehdä [51, s. 8], mutta muutamia oletuksia voidaan tehdä. Perinteiset menetelmät ovat suunnitteluvetoisia toisin kuin ketterät menetelmät, jotka nähdään reaktiivisina. Ketterät menetelmät korostavat ihmisläheisyyttä, yhteistyötä ja ihmisten välistä viestintää. Perinteiset menetelmät puolestaan nojaavat enemmän järjestelmällisyyteen, ohjeistamiseen ja suunnitelmallisuuteen [59, s. 68]. Perinteiset menetelmät olettavat ohjelmistokehityksen ennustettavaksi ja täysin määriteltäviksi, jolloin huolellisella suunnittelulla voidaan saavuttaa projektille määritellyt tavoitteet. Ketterät menetelmät taas olettavat, että muutoksilta ei voida välttyä. Muutoksiin varaudutaan ja niihin reagoidaan nopeasti [59, s. 68]. Boehm kuvaa perinteisten ja ketterien menetelmien suunnitelmallisuuden kirjoa kuvan 2.1 mukaisesti. Äärivasemmalla on kuritonta suunnittelemattomuutta ja oikealla pilkuntarkkaa hallinnointia.



Kuva 2.1: Suunnitelmallisuuden kirjo [59, s. 65].

Ketterät menetelmät ovat iteroivia ja inkrementaalisia, kuten myös perinteiseksi malliksi luettava spiraalimalli. Craig Larman ja Vitor Basili toteavat iteratiivisten

ja inkrementaalisten kehitystapojen historiasta kertovassa artikkelissaan [64] suoraviivaisten dokumenttivetoisten mallien olevan helpommin omaksuttavia, koska ne ovat yksinkertaisia, mitattavia ja niihin viitataan paljon ohjelmistotuotannon kirjallisuudesta. Heidän mielestään idea näistä malleista tulisi hylätä ja onnistuneen ohjelmistokehityksen nimissä suosia iteratiivisia ja inkrementaalisia kehitystapoja.

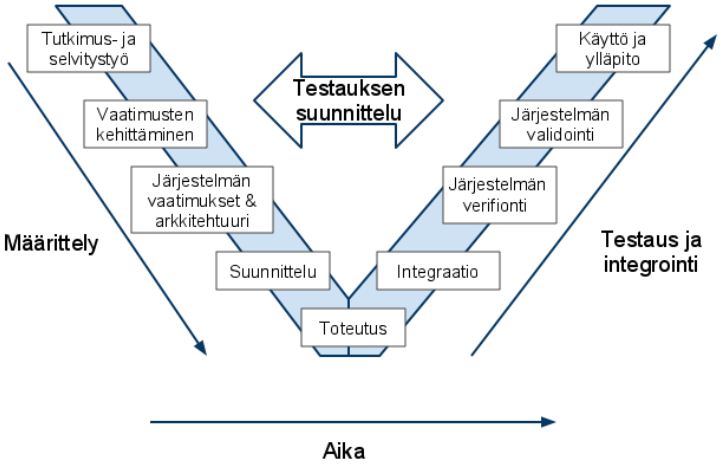
Ohjelmistotuotannon käsitys, erityisesti metriikoiden valvominen, on saanut myöhemmin kritiikkiä Tom DeMarcolta, josta hän kirjoittaa artikkelissa *"Software engineering: An Idea Whose Time Has Come and Gone?"* [55]. Samaisessa artikkelissa DeMarco neuvoo lähestymään ohjelmistokehityksen hallinnoimista ketterien menetelmien inkrementaalisesta näkökulmasta. Hän näkee ohjelmistokehityksen kannalta tärkeimpänä tavoitteena muutoksen vallalla olevista käytänteistä. Ohjelmistojen kehitys on DeMarcon mukaan kokeellista.

2.2 V-malli

V-malli esitettiin Saksassa ja se on tällä hetkellä vaadittu kehitysmalli kaikissa maan hallituksen IT-hankkeissa [25], [28]. Samainen malli on myös yleisesti käytössä Yhdysvaltojen Department of Defence ja Department of transportation -laitoksilla [26], [27]. Malli on myös kansainvälisesti hyväksytty eri standardoimisorganisaatioiden toimesta ja siihen viitataan eri standardeista kuten ISO/IEC 12207:sta ja ISO 9001:sta [25].

Kuvassa 2.2 on esitetty V-mallin vaiheet. Vaatimusmäärittely alkaa tutkimus- ja selvitystyöllä, minkä aikana arvioidaan eri ratkaisujen taloudellinen, tekninen ja soveltuvuus. Teknisiin toteutuksiin otetaan kantaa vain sen verran, että saadaan yleinen käsitys toteutustavasta. Saatava hyöty ja kustannukset arvioidaan samalla kun tunnistetaan projektiin liittyvät riskit. Tutkimusvaiheessa kehotetaan kartoittamaan muutamia eri toteutustapoja, joista valitaan soveltuvin erikseen määritellyillä mittareilla. Vaiheen tuloksena syntyy dokumentaatio aiheen kontekstista ja ehdotetusta ratkaisusta.

Toisessa vaiheessa tunnistetaan ja kuvataan järjestelmän vaatimukset korkealla tasolla ymmärrettävässä muodossa — vastaten kysymyksiin kuka, mitä, miksi, missä ja miten järjestelmässä. Vaiheen lähtötietoina on ensimmäisessä vaiheessa kuvattu ratkaisuehdotus. Kaikki osapuolet, joita järjestelmä koskettaa, tunnistetaan ja käyttäjäroolit kuvataan. Jokaisen osapuolen tarpeet kuvataan vaatimuksina, joten jokainen vaatimus kuuluu aina jollekin tai useammalle osapuolelle. Vaiheen tuloksena syntyy toiminnallisuuden kuvaus (*engl. Concept of operations*). Lopuksi kun järjestelmän korkean tason vaatimuksista on riittävä ymmärrys, tehdään järjestelmän



Kuva 2.2: Yksinkertaistettu kuvaus V-mallista.

alustava validointisuunnitelma laatuattribuuttien avulla.

Kolmannen vaiheen suunnittelun lähtökohdaksi otetaan korkean tason vaatimusmäärittelyn ja haetaan vastaus kysymykseen: *mitä järjestelmä tekee*? Tässä vaiheessa työskennellään vielä läheisesti kaikkien osapuolien kanssa ja vaatimuksia yritetään saada esille eri menetelmillä osapuolien kanssa. Tavoitteena on tuottaa validoitu joukko järjestelmävaatimuksia ja näihin perustuen järjestelmän verifiointi- sekä hyväksymissuunnitelma.

Suunnitteluvaiheessa järjestelmävaatimuksista johdetaan tarkemmat vaatimukset. Tämä vaiheen lähtökohdaksi otetaan kaikki aikaisempien vaiheiden tulokset. Aikaisemmin kuvattu kokonaisjärjestelmä pilkotaan alijärjestelmiksi ja lopulliset vaatimukset jaetaan tunnistettuihin alijärjestelmiin. Vasta tässä vaiheessa tutkitaan tekniikka ja mahdolliset kolmannen osapuolen järjestelmät tarkemmin. Tunnistetut tekniikat ja muut järjestelmät arvioidaan ja soveltuvin valitaan. Laadullisiin ja muihin sitoviin vaatimuksiin — kuten standardeihin ja säädöksiin — kiinnitetään tarkempaa huomiota suunnittelussa. Suunnitteluvaiheen lopussa kehitetään alijärjestelmille ja ohjelmistokomponentteja koskevat verifiointi- ja hyväksymissuunnitelmat sekä integrointisuunnitelma. Vaiheen tulokset tukevat kehityksen aikana tehtävää yksityiskohtaista suunnittelua.

V-mallissa korostuu laadunhallintaan liittyvät tehtävät. Ne on kuvattu mallissa verifiointi- ja validointitehtävinä, joita määritellään suunnittelun aikana. Verifiointia ja validointia toteutetaan jo vaatimusmäärittelyn ja suunnittelun aikana, sillä kehityksen alkuvaiheessa virheiden korjaaminen on huomattavasti halvempaa kuin myöhemmin [65, s. 76]. Ohjelmiston lopullinen validointi suoritetaan kehityksen lopussa.

V-mallia on moitittu siitä, että se on hyvin paljon samankaltainen kuin vesiputousmalli. Esitetyssä mallissa suunnittelua seuraa toteutus, integraatio ja testaus. Käytännössä V-malli on ”alakolmiossa” iteroiva, joten tarkempaa suunnittelua, kehitystä ja integrointia tehdään useita kertoja ennen kuin järjestelmä on valmis lopulliseen testaukseen. Malliin viitataan testauksessa usein testauksen V-mallina, johtuen eri vaatimusmäärittelyvaiheiden aikana syntyviin alustaviin testaus- ja verifiointisuunnitelmiin. Sinällään esitelty malli ei ole testausmalli.

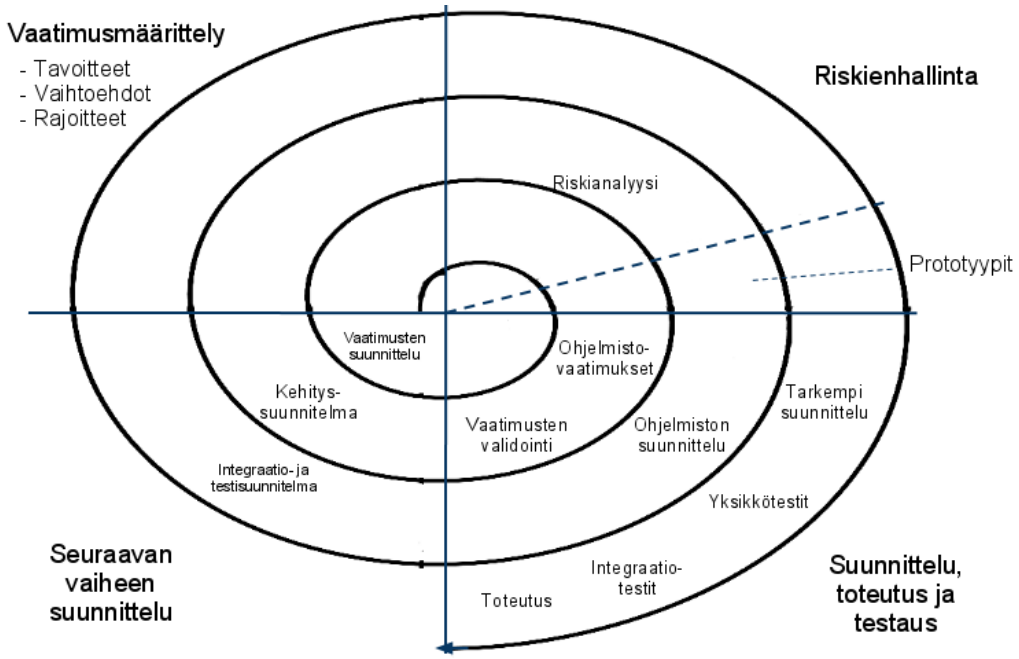
V-malli on luonteeltaan hyvin suunnitelma- ja dokumenttivetoinen ohjelmistokehitysmalli.

2.3 Spiraalimalli

Barry Boehm esitteli spiraalimallin artikkelissaan *”A Spiral Model of Software Development and Enhancement”* [29] vuonna 1988. Spiraalimallissa järjestelmää kehitetään iteratiivisesti tehden protomalleja kehityksen aikana tuotteesta. Ensimmäisten iteraation jälkeen tuote voi olla hyvinkin yksinkertainen — jopa pelkkä hahmotelma paperilla. Iteraation jälkeen edellisen iteraation tuotos arvioidaan, jonka pohjalta aloitetaan seuraava iteraatio. Mallia käytettäessä ohjelmisto valmistuu inkrementaalisten julkaisujen myötä. Spiraalimallin yksi tärkeimmistä piirteistä on sen riskivetoisuus (*engl. risk-driven*) [67, s. 3]. Malli on esitetty kuvassa 2.3 ja sen vaiheet ovat

1. Vaatimusmäärittely
2. Alustava suunnittelu ja riskianalyysi
3. Toteutus ja testaus
4. Seuraavan iteraation suunnittelu

Vaatimusmäärittelyvaiheessa järjestelmän vaatimukset tunnistetaan tarkasti tulevan iteraation osalta. Tässä vaiheessa käytetään haastatteluita tai muita esillesaamistekniikoita kaikkien eri osapuolien kanssa. Myös vaihtoehtoiset toteutustavat ja rajoitteet tunnistetaan. Seuraavaksi arvioidaan eri toteutusvaihtoehtojen soveltuvuus rajoitteiden ja vaatimusten suhteen. Eri vaihtoehdoille tunnistetaan riskejä, jotka saattavat johtaa projektin epäonnistumiseen. Riskien lähteiden tunnistamiseksi voidaan käyttää simulointia, tutkia vastaavanlaisia projekteja tai haastatella käyttäjiä. Malli ehdottaa prototyypin tekemistä, koska se on suhteellisen riskitön ja halpa tapa kuvata osaa tulevasta järjestelmästä. Prototyypin toteutuksen ja testauksen



Kuva 2.3: Spiraalimalli.

jälkeen viimeisenä vaiheena tehty iteraatio ja sen tuloksena syntynyt prototyyppi tai muu tuotos arvioidaan. Arvion jälkeen määritellään seuraavan iteraation vaatimukset, suunnitellaan ja toteutetaan seuraava inkrementti ohjelmistosta.

Spiraalimalli määrittelee melko löyhästi kuinka ja millä tarkkuustasolla vaatimukset kuvataan ensimmäisen iteraation aikana. Boehm kirjoittaa alkuperäisessä artikkelissa mallin tulevan yhdenvertaiseksi malliksi (perinteisten) mallien kanssa, kun järjestelmän vaatimukset ovat vakaita, eivätkä muutu kehityksen aikana. Malli ei ota kantaa kuinka kauan yksi iteraatio kestää tai kuinka monta iteraatiota tarvitaan valmiin ohjelmiston kehittämiseen. Boehmin artikkelin aikana toteutetussa projektissa ensimmäiseen iteraatioon käytettiin kaksi miestyökuukautta ja toiseen jopa 12 miestyökuukautta. Toisaalta Boehm kirjoittaa mallin tarjoavan riskivetoisuutensa vuoksi sisäisesti työkalun vaatimusten analyysiin, suunnitteluun ja testaukseen käytettävän ajan estimoimiseen.

Spiraalimallia voi pitää realistisena lähtökohtana suurten tietojärjestelmien kehittämiseen riskivetoisuutensa vuoksi [66, s. 38]. Riskit ovat tunnistettavissa ohjelmiston kehityksen aikana ja mikäli jokin riski todetaan liian suureksi, voidaan kehitys lakkauttaa tai vaatimuksia muuttaa vielä ennen kuin on projekti ylittää sille asetetut resurssit tai rajoitteet.

Mallin ymmärtämisessä ja käytössä on ollut haasteita. Spiraalimalli on ymmärretty vain sarjaksi peräkkäisiä vesiputousmalleja, joissa jokainen vaihe on suoritettava eikä vaiheiden välillä ole kytköstä edelliseen [67, s. 3]. Mallia on myöhemmin paranneltu ja vaiheita on tarkennettu. Boehmin WinWin-spiraalimallissa on perinteisen mallin lisäksi määritelty tarkemmin käyttäjien ja osapuolten odotusten huomiointi [1] ja iteraatioiden edistystä seuraavia näkymiä [68, s. 34].

2.4 Yhteenveto

Ohjelmistojen kehityksen tavoite on saada aikaiseksi tuote, joka täyttää asiakkaan odotukset sitä kohtaan. Mallit ja menetelmät antavat tekemiseen työkaluja ja auttavat hahmottamaan kehityksen aikaisia asioita. On kuitenkin muistettava, että ohjelmiston tekee edelleen ihminen. Mallit ovat vain opastamassa tekemistä. Pressman vertaa luovaa ohjelmistosuunnittelijaa maalaajaan, joka nauttii jokaisesta pensselin vedosta yhtä paljon kuin valmiista taulusta [66, s. 47]. Ohjelmistokehitysmallien tulisi suoda suunnittelijalle vapaus nauttia työstään. Huono malli johtaa helposti kelvottomaan lopputulokseen.

[1] Jokaisen osapuolen välillä yritetään löytää ”win-win” –tilanne, mistä myös mallin nimi tulee.

3 Scrum

Scrum on ketterä ohjelmistokehitysmalli, joka kuvaa verrattain vähän tehtäviä ja rooleja. Ketterien menetelmien arvot on kirjattu Agile Manifestossa [19]. Agile Manifeston esittelivät mm. sellaisten metodologien kuten Extreme Programming, Scrum ja Feature Driven Development puolestapuhujat. Agile manifeston sisältö suomennettuna on seuraava:

> Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan:
>
> **Yksilöitä ja vuorovaikutusta** enemmän kuin prosesseja ja työkaluja
> **toimivaa sovellusta** enemmän kuin kokonaisvaltaista dokumentaatiota
> **asiakasyhteistyötä** enemmän kuin sopimusneuvotteluita ja
> **muutokseen reagoimista** enemmän kuin suunnitelman noudattamista.
>
> Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia asioita enemmän.

3.1 Yleiskuvaus

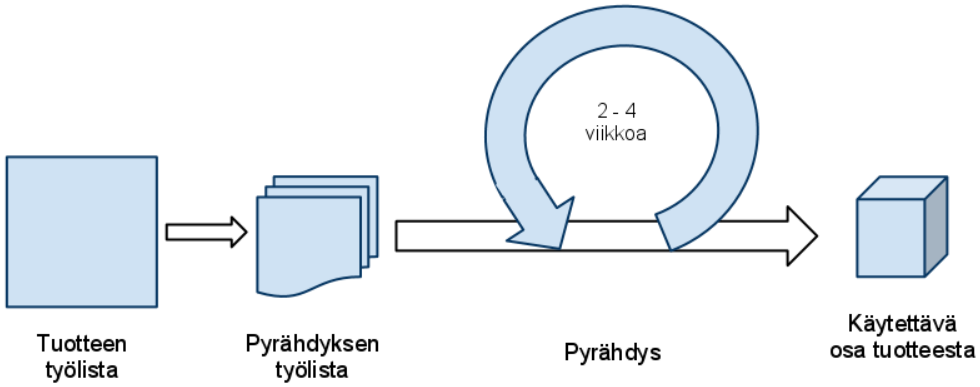
Scrum-termi on otettu rugbystä, jossa se tarkoittaa pelin aloitusryhmitystä. Kirjallisuudessa termi esiteltiin tiettävästi ensimmäisen kerran Hirotaka Takeuchin ja Ikujiro Nonakan vuonna 1986 julkaisemassa artikkelissa ja sillä kuvataan Japanissa käytettyä tuotekehitysprosessia, joka on nopea, joustava ja itseohjautuva [8, s. 1]. Ken Schwaber esitteli tässä tutkielmassa kuvatun Scrum-mallin artikkelissa *SCRUM Development Process* vuonna 1995 [9].

Scrum on iteratiivinen ja inkrementaalinen ohjelmistokehitysmalli kuten aikaisemmin esitelty spiraalimalli. Iteratiivisten vaiheiden tueksi Scrum määrittelee toimintatapoja ja rooleja. Iteraatioita kutsutaan Scrum-mallissa pyrähdyksiksi (*sprint*) [9, s. 10] ja ne ovat viikon tai korkeintaan neljän viikon mittaisia [9, s. 14]. Jokaisen pyrähdyksen tavoitteena on julkaista toimiva ohjelmisto niillä ominaisuuksilla, jotka valittiin toteutettaviksi [8, s. 30]. Pyrähdykset seuraavat toisiaan kunnes ohjelmisto päätetään julkaista käytettävissä olevien resurssien tai toteutettujen asiakas-

vaatimuksien perusteella [9, s. 14].

Kuvassa 3.1 on esitetty mallin kannalta olennaisimmat asiat. Scrum-mallin tärkein osa on pyrähdys, jonka aikana kehitys tapahtuu. Tätä edeltää palaveri, jossa valitaan kyseisen pyrähdyksen aikana toteutettavat vaatimukset. Scrum-mallin toteuttava tekijä on tiimi, jota opastaa ja valvoo Scrum-mestari (*Scrum Master*). Pyrähdystä edeltävässä palaverissa tuotteen omistaja (*Product owner*), Scrum-mestari ja tiimi sekä mahdolliset muut osapuolet kokoontuvat ja valitsevat seuraavan pyrähdyksen aikana toteutettavat vaatimukset. Tiimillä on palaverissa tärkeä osa – tiimi ja tuotteen omistaja valitsevat yhdessä sellaiset ominaisuudet, jotka on mahdollista toteuttaa pyrähdyksen aikana. Pyrähdykselle asetetaan tavoite ja tiimi sitoutuu tähän. Pyrähdyksen aikana tiimi suorittaa kehitystyön itsenäisesti, eikä sovittuun pyrähdykseen lisätä uusia tehtäviä.

Yleensä jokaisen pyrähdyksen tavoitteena on saada asiakkaalle jotain näkyvää ja toimivaa [8, s. 30]. Kehitystyön alussa toiminnallisuus voi olla – ja usein onkin – todella yksinkertaista. Tarkoitus on kuitenkin sitouttaa tuotteen tilaaja kehitykseen mukaan ja näin ollen saada jatkuvaa palautetta. Tällä tavoin varmistetaan, että tehdään oikeaa asiaa ja pienennetään perinteisiä ohjelmistokehityksen riskejä.



Kuva 3.1: Scrum [36, s. 5].

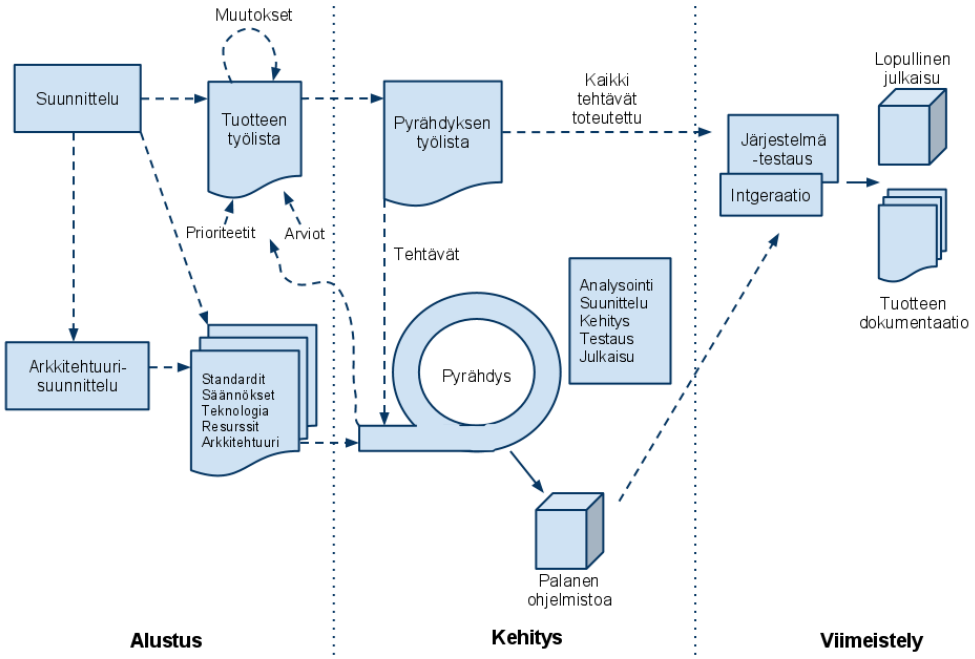
Pyrähdyksen aikana tiimi suunnittele, toteuttaa ja testaa ohjelmiston. Tiimi kommunikoi keskenään päivittäisissä Scrum-tapaamisissa, joiden tarkoituksena on varmistaa, että jokainen tiimin jäsen pääsee ilmaisemaan työn kulustaan ja mahdollisista ongelmistaan [8, s. 40]. Yleensä tässä tapaamisessa seurataan myös pyrähdyksen pysymistä aikataulussa. Jokaisen pyrähdyksen jälkeen on uusi tapaaminen, jossa esitellään saavutetut tulokset. Scrum-mallissa kehotetaan pitämään tuotteen omistajalle ja muille mahdollisille osapuolille demonstraatio tehdystä ohjelmistosta kehitysympäristössä. Tämän demonstraation aikana paikalla olevat voivat antaa

palautetta ja kehitysideoita.

Jokaisen pyrähdyksen jälkeen pidetään myös palaveri tiimin ja Scrum-mestarin kesken. Palaverin tarkoitus on seurata, parantaa ja mahdollisesti muuttaa prosessia. Tämä on yksi tärkeimmistä palavereista Scrum-mallin toiminnan kannalta.

3.2 Mallin vaiheet

Ken Schwaber jakaa Scrum-mallin kolmeen vaiheeseen vuonna 1995 julkaistussa *SCRUM Development Process* -artikkelissa. Vaiheet ovat: alustus (*pre-game*), kehitys (*game*) ja viimeistely (*post-game*) [9, s. 12]. Muissa julkaisuissa tätä vaihejakoa ei korosteta yhtä paljon vaan niissä keskitytään enemmän kehitysvaiheeseen ja sitä tukeviin tehtäviin. Vaihejako ja niiden tehtävät on esitetty kuvassa 3.2.



Kuva 3.2: Mallin vaiheet [9, s. 10, 12].

Alustusvaihe koostuu kahdesta tehtävästä: suunnittelusta ja järjestelmän arkkitehtuurin kuvaamisesta. Suunnittelun aikana kehitetään tuotteen työlista niin kattavasti kuin mahdollista. Työlistan perusteella arvioidaan julkaisuaikataulu ja mahdolliset osajulkaisut määritellään sekä arvioidaan tarvittavat resurssit. Samoin arvioidaan riskit ja sopivat riskikontrollit otetaan käyttöön [9, s. 13]. Riskienhallintaa ei tarkemmin määritellä ja riskit koskevat projektiriskejä. Arkkitehtuuri ja korkean tason suunnittelu tehdään tuotteen työlistan pohjalta. Arkkitehtuuri arvioidaan ja siinä otetaan huomioon mahdolliset haasteet.

Kehitysvaihe on Scrum-mallin iteratiivinen vaihe. Kehitys koostuu pienemmistä vaiheista, jotka ovat pyrähdyksien suunnittelu- ja katselmointipalaverit, tiimin katselmoinnit ja itse pyrähdykset. Pyrähdyksiä suoritetaan kunnes ohjelmisto todetaan valmiiksi julkaistavaksi. Vaatimusten muuttuminen on oletettua ja muutosta pyritään hallitsemaan pyrähdysten aikana Scrum-käytäntöjen avulla. Ongelmia tai muutoksia hallitaan esimerkiksi päivittämällä työlistaa jokaisen pyrähdyksen välillä tai muuttamalla julkaisuja.

Viimeistelyvaiheeseen siirrytään kun ohjelmisto on todettu valmiiksi julkaistavaksi, joko toteutettujen asiakasvaatimusten perusteella tai resurssien puitteissa. Uusia ominaisuuksia ei enää toteuteta. Vaiheessa tehdään julkaisun vaatimat työt kuten viimeiset integraatiot, järjestelmän julkaisutestaus ja kirjoitetaan ohjelmistoon liittyvä dokumentaatio.

3.3 Roolit

Scrum-mallin rooleista on käytetty termiä "The Chicken and the Pig" [8, s. 42], [13], mutta tämä jaottelu on saanut myöhemmin arvostelua kohteekseen [12]. Nimitys on seurausta siitä, kuinka roolit ovat eriarvoisia sen suhteen millä tavoin ne sitoutuvat kehitysprosessiin.

Possuroolit, eli roolit, jotka sitoutuvat täysin projektiin ja ovat vastuussa tuloksista:

- **Tuotteen omistaja** (*Product owner*) on henkilö, joka edustaa projektissa asiakasta. Hänen vastuullaan on tuote, tuotteen työlistan ylläpito ja hän vastaa siitä, että projektissä tehdään oikeita asioita.

- **Scrum-mestari** (*Scrum Master*) hallitsee Scrumia ja vastaa siitä, että tiimi pääsee työskentelemään estoitta. Scrum-mestari ei ole tiimin johtaja tai projektipäällikkö, mutta hänen vastuullaan on varmistaa, että Scrumia käytetään, kuten sitä on tarkoitettu. Scrum-mestari on avainasemassa mallin toiminnan kannalta. Hän järjestää palaverit, opettaa Scrum-mallin toimintatavan tuotteen omistajalle sekä toimii yhteyshenkilönä asiakkaan ja tiimin välillä. Tämä ei kuitenkaan tarkoita sitä, ettei asiakas voisi olla suoraan yhteydessä tiimin jäseniin.

- **Tiimi** koostuu useammasta henkilöstä, useimmiten kahdesta yhdeksään henkeä. Tiimin vastuulla on toimittaa ohjelmisto aikataulussa. Tiimi koostuu useista osaajista, jotta ohjelmisto saadaan kehitettyä. Tiimin sisällä ei usein ole

jakoa erikseen testaajien, käyttöliittymäsuunnittelijoiden tai muiden erikoisosaamisen mukaan vaan tiimi jakaa tehtäviä yhteisesti. Tiimit ovat itsestään organisoituvia. Tiimi saa itse päättää kuinka valitut ominaisuudet tehdään. Tiimit pysyvät samana pyrähdysten aikana, mutta pyrähdysten välissä tiimin henkilöitä voi vaihtaa - vaihtamisella on tosin vaikutusta tiimin tuottavuuteen.

Kanaroolit, eli roolit, jotka opastavat projektia ja kuulevat projektin etenemisestä:

- **Asiakkaat, toimittajat ja kauppiaat** muodostavat ohjelmiston tarpeen ja näin mahdollistavat ohjelmistokehityksen. Tämä sidosryhmä ei ota kantaa ohjelmiston kehitykseen muussa vaiheessa kuin pyrähdyksien vaihdon yhteydessä.
- **Johto** mahdollistaa ohjelmistokehityksen yhtiössä.

3.4 Tapaamiset

Julkaisusuunnittelupalaveri (*Release planning meeting*). Projektin alussa tehdään julkaisusuunnitelma ja asetetaan yhteiset päämäärät. Tarkoituksena on saada aikaan yhteisymmärrys tuotteesta ja tuotteen tärkeimmistä ominaisuuksista, joita julkaistava tai julkaistavat versiot sisältävät. Yleensä yrityksillä on jo tapa tehdä julkaisusuunnitelma, jossa lukitaan tärkeitä ominaisuuksia ja päivämääriä. Scrum-mallin ja ketterien menetelmien kannalta tämä saattaa olla haastavaa, koska vaatimusmäärittely ja prioriteetit muuttuvat jatkuvasti. Julkaisusuunnitelman tarkoitus onkin Scrum-mallissa saada priorisoitua tärkeät ominaisuudet ja toiminnallisuudet sen hetkisestä työlistasta. Tämän avulla tuotteen kehityksen edistymistä voidaan seurata.

Päivittäinen tapaaminen (*Daily Scrum*). Tiimit tapaavat päivittäin viidentoista minuutin ajan. Tämä tapaaminen tapahtuu aina samassa paikassa, samaan aikaan ja silloin vastataan lyhyesti kolmeen kysymykseen: Mitä olen tehnyt viimeksi? Mitä aion tehdä ennen seuraavaa tapaamista? Mikä hidastaa minua? Tarkoituksena on parantaa tiimin välistä tiedonkulkua, välttää turhia palavereja ja huomata sekä poistaa häiriöitä pyrähdyksen aikana.

Tapaamisen järjestäminen on Scrum-mestarin vastuulla ja hän valvoo, että tapaaminen pysyy ajallaan ja ettei tapaamisen asiasisältö rönsyile. Tapaamisen aikana äänessä saa olla vain tiimi. Usein tapaamisen aikana tarkastetaan onko pyrähdys aikataulussa. Tällöin päivitetään pyrähdyksen työkaaviota (*sprint burndown*).

Pyrähdyksen katselmointipalaveri (*Sprint review meeting*). Tämä tapaaminen pidetään jokaisen pyrähdyksen lopussa. Tällöin tarkistetaan pyrähdyksen tulokset.

Tapaamisessa ovat paikalla tiimi, Scrum-mestari, tuotteen omistajat ja muut asianomaiset. Tiimi esittää, mitä tehtiin, mitä ei tehty ja saavutettiinko pyrähdyksen tavoite. Tapaamisessa näytetään tehty ohjelmisto oikeassa ajoympäristössä – usein suoraan kehityskoneella [8, s. 30]. Tapaamisen aikana tiimi vastaa ohjelmistoa koskeviin kysymyksiin. Tapaamisen tarkoituksena on esittää saavutetut tulokset ja kerätä tietoa seuraavaan pyrähdyksen suunnittelupalaveriin.

Pyrähdyksen suunnittelupalaveri (*Sprint planning meeting*). Tämä tapaaminen jakautuu kahteen osaan. Ensiksi vastataan kysymykseen ”mitä” ja seuraavaksi ”miten”. Tapaamisessa sovitaan seuraavan pyrähdyksen tavoitteet. Siihen saavat osallistua kaikki. Tuotteen omistaja esittää tuotteen työlistasta (*product backlog*) vaatimukset, jotka on priorisoitu aikaisemmin. Tiimi vastaa siitä, että pyrähdykseen valittu työmäärä on tehtävissä sovitussa aikataulussa. Kun sopivat toiminnallisuudet on löydetty, valitaan ne uuteen pyrähdykseen, josta muodostuu pyrähdyksen työlista (*sprint backlog*). Pyrähdyksen työlistan koko on riippuvainen tiimistä ja ainoastaan tiimi voi päättää kuinka paljon tehtäviä jokaiseen pyrähdykseen voidaan valita.

Pyrähdyksen työlistan perusteella pyrähdykselle tehdään tavoite. Tämä tavoite opastaa tiimiä ja antaa tiimille hieman vapautta tekemisessä. Ongelmatilanteessa voidaan tarkistaa pyrähdyksen tavoite ja priorisoida tekemisiä tavoitteen suhteen. Samoin tämä on kysymys, johon vastataan pyrähdyksen katselmointitapaamisen aikana: Saavutettiinko tavoite?

Kun pyrähdykselle on valittu tavoite ja toteuttavat ominaisuudet, alkaa tiimi suunnitella kyseistä pyrähdystä. Valitut ominaisuudet pilkotaan pienemmiksi tehtäviksi. Tehtävät yritetään jakaa toisistaan riippumattomiksi ja korkeintaan yhden päivän mittaisiksi. Tehtävät ovat konkreettisia asioita, joita jokainen tiimin jäsen tekee pyrähdyksen aikana.

Tuotteen omistaja on yleensä paikalla palaverin toisessa osassa, jossa tiimi pilkkoo pyrähdyksen tehtävät pienemmiksi tehtäviksi. Tällöin häneltä voidaan kysyä tarkentavia kysymyksiä liittyen tehtäviin ominaisuuksiin.

Jälkitarkastelu (*Sprint retrospective*). Jokaisen pyrähdyksen jälkeen pidetään palaveri, johon osallistuu tiimi ja Scrum-mestari. Tarkoituksena on parantaa prosessia. Palaverin aikana vastataan kysymyksiin: mikä meni hyvin ja missä on parannettavaa? Palaverin tuloksena syntyy konkreettisia asioita, joita kokeillaan seuraavan pyrähdyksen aikana.

3.5 Scrum-mallin työkalut

- **Tuotteen työlista** (*Product backlog*) on tuotteen vaatimusmäärittely. Tuotteen omistaja ylläpitää dokumenttia, halliten sen sisältöä, saatavuutta ja priorisointia. Tiimi tekee tehtävien työmäärän arvionnin. Dokumentti elää jatkuvasti: sen sisältö muokkautuu, vaatimuksia lisätään tai poistetaan ja prioriteetit muuttuvat. Vaatimukset on järjestetty prioriteettien mukaan. Tähän dokumenttiin on listattu kaikki tarpeelliset ominaisuudet, jotka täytyy toteuttaa, jotta ohjelmisto voidaan onnistuneesti julkaista. Se on lista toiminnallisuuksista, ominaisuuksista, parannuksista ja muista tarpeista.
- **Pyrähdyksen työlista** (*Sprint backlog*) sisältää pyrähdykseen valitut toiminnallisuudet, jotka on poimittu tuotteen työlistasta pyrähdyksen suunnittelupalaverin aikana.
- **Julkaisun työkaavio** (*Release burndown*) on graafi, jonka avulla seurataan tuotteen aikataulussa pysymistä.
- **Pyrähdyksen työkaavio** (*Sprint burndown*) on graafi, jonka avulla seurataan pyrähdyksen aikataulussa pysymistä.

3.6 Yhteenveto

Scrum on iteratiivinen ja inkrementaalinen ohjelmistokehityksen projektinhallintamalli. Se lähestyy ohjelmistokehitystä empiirisestä näkökulmasta ja soveltaa ideoita teollisuuden prosessikontrolliteorioista [8, s. 1-2]. Mallin tavoitteena on pitää ohjelmistokehitys joustavana, mukautuvana ja tuottavana. Scrum ei määrittele suoraan ohjelmistokehitystekniikoita, vaan keskittyy siihen, kuinka tiimin tulisi toimia muuttuvassa kehitysympäristössä [51, s. 27]. Mallilla on muutamia tunnusomaisia piirteitä, kuten Schwaberin esittämät [9, s. 17] ominaisuudet:

- Joustava julkaisu — julkaisun sisältö määräytyy ympäristön mukaan. Ympäristö koostuu julkaistavaa tuotetta koskevista rajoitteista, kuten kohdealustasta, standardeista tai resursseista.
- Joustava aikataulu — julkaisu saattaa tapahtua aikaisemmin kuin suunniteltiin.
- Pienet tiimi — jokaisessa tiimissä on maksimissaan kuusi henkeä ja yhdessä projektissa voi olla useampi tiimi.

- Toistuvat katselmoinnit — tiimin edistymistä seurataan päivittäin ja pyrähdyksittäin. Jokaiseen katselmointiin toimitetaan toimiva ohjelmiston palanen.
- Yhteistyö — projektilta odotetaan sisäistä ja ulkoista yhteistyötä.

Hernik Kniberg puolestaan esittää Scrum-mallin tärkeimmiksi toiminnoiksi pyrähdyksen suunnittelupalaverin ja jälkitarkastelut [53, s. 77]. Näiden lisäksi Kniberg esittää [52] Scrum-mallin peruselementeiksi tuotteen ja pyrähdyksen työlistat, aikarajatut pyrähdykset, selkeästi määritellyt Scrum-roolit, ohjelmiston esittämisen oikeassa ajoympäristössä sekä selkeästi määritellyn valmiuskriteerin (*Definition of Done*). Ohjelmistokehitysmallista tulee Scrum kun edellä esitetyt ominaisuudet täyttyvät.

4 Vaatimusmäärittely ja vaatimustenhallinta

Vaatimusmäärittely on nostettu erilliseksi luvuksi tässä tutkielmassa johtuen sen tärkeästä roolista lääkinnällisen laitteen ohjelmistokehityksessä. Tämän vaiheen aikana tunnistetaan ja analysoidaan ohjelmistoa koskevat lait ja säännökset, jotka ohjaavat ja osaltaan rajoittavat lääkinnällisen laitteen ohjelmistokehitystä [11, s. 9-12].

Tutkimusten mukaan puutteelliset vaatimukset ja käyttäjien huono sitoutuminen projektiin ovat merkittävimpiä tekijöitä projektien epäonnistumisessa [61], [62, s. 245-246]. Vaatimusmäärittely on haasteellista, koska ohjelmistot ovat kompleksisia, käyttäjä ei välttämättä osaa ilmaista odotuksiaan ja ohjelmiston käsitys on käyttäjille abstrakti.

Puutteellisten vaatimusten, parhaiden käytänteiden epäkypsyyden ja uuden teknologian johdosta ohjelmistokehitys on luonteeltaan tutkimustyötä, jonka seurauksena kehitys toimii suunnitteluna. Näiden ongelmien myötä muutokset vaatimuksiin ovat väistämättömiä [60, s. 8-12]. Onnistunutta vaatimusmäärittelyä ja vaatimustenhallintaa voidaan näin ollen pitää ohjelmistokehitysprojektin onnistumisen näkökulmasta kriittisimpänä toimintona.

4.1 Esitutkimus

Esitutkimus on prosessi, jonka avulla tunnistetaan esitettyyn projektiin liittyvät ongelmat ja mahdollisuudet. Vaihe edeltää varsinaista teknistä vaatimusmäärittelyä ja projektin aloitusta [69, s. 185]. Esitutkimuksessa tunnistetaan onko projekti toteuttamiskelpoinen ja kuvataan projektia koskevat rajoitteet, riskit ja kustannukset. Tutkimuksen avuksi on erilaisia analyysitekniikoita, kuten SWOT (*Strengths, Weaknesses, Opportunities, Threats*) [70], PEST (*Political, Economic, Social, and Technological Analysis*) ja TELOS (*Technology and System, Economic, Legal, Operational, Schedule*) [69, s. 186]. TELOS-analyysi kattaa seuraavat alueet:

- **Teknologian ja järjestelmän soveltuvuus** -analyysi käsittelee onko projektille sopivaa teknologiaa olemassa ja osataanko sitä soveltaa. Analyysin tuloksena esitellään mitä teknologiaa voidaan soveltaa ja mitä vaatimuksia tämä asettaa projektin läpiviennille.
- **Taloudellinen soveltuvuus** -analyysissa arvioidaan esitetyn projektin kustan-

nukset ja hyödyt esitettyihin resursseihin, liiketoimimahdollisuuksiin ja ratkaisuihin perustuen.

- **Lainmukaisuus**-analyysi tutkii koskeeko projektin sovellusta jokin lainsäädäntö. Projektia koskevia säännöksiä ja lakeja voi esiintyä eri maiden lainsäädännössä tai organisaation omissa ohjeissa. Tunnistetut säännökset kirjataan ja niiden vaikutus projektiin arvioidaan. Erityisesti lääkinnällisten laitteiden kehityksessä on tärkeätä tunnistaa laitetta koskevat lainsäädäntö ja määräykset projektin alkuvaiheessa, ennen tarkempaa toiminnallisuuden määrittelyä [11, s. 9].

- **Toiminnallisuuden soveltuvuus** -analyysissä arvioidaan kuinka hyvin esitetty projekti soveltuu aiottuun käyttötarkoitukseen. Näkökulmaksi otetaan aikaisemmin määritelty ongelma, jonka projektin tuotos ratkaisee ja mikä rahallinen tai muu mitattava hyöty toiminnallisuudella saavutetaan.

- **Aikataulutus**-analyysissä arvioidaan millä aikataululla projekti viedään läpi ja onko aikatauluarvio realistinen resursseihin tai muihin rajoittaviin tekijöihin nähden. Aikatauluanalyysin perusteella säädetään aikataulua ja resursseja kunnes ne realistisesti mahdollistavat projektin.

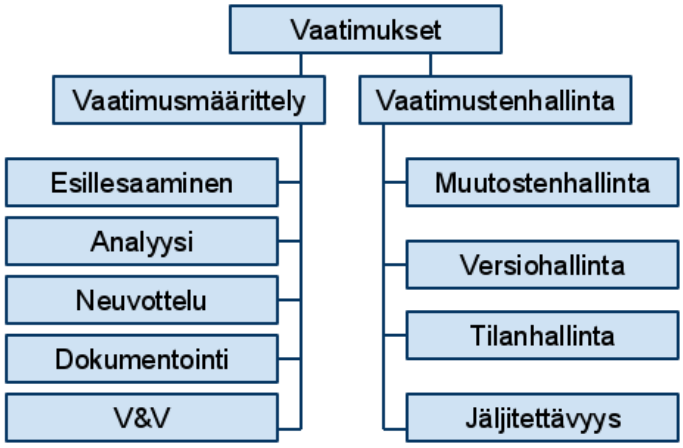
Edellä esitetyt analyysimetodit ovat laadullisia tapoja tutkia projektin ominaisuuksia ja haasteita. Niitä voidaan pitää luonteeltaan aivoriihimäisinä ja niiden onnistunut käyttö edellyttää asiantuntijuutta. Wiegersin mukaan tämän vaiheen tärkeimmät ohjaavat tekijät ovat tuotteen visio ja liiketoiminnan vaatimukset, jonka perusteella projektin laajuus, konteksti ja rajoitteet tunnistetaan. [34, l. 5].

Esitutkimuksen tuloksena syntyy toteutettavalle projektille arvokasta lisätietoa, jota voidaan käyttää suunnittelun apuna. Erityisesti vaatimusmäärittely hyötyy vaiheen tuloksista ja ne voidaan sellaisenaan ottaa vaatimusmäärittelyn esitiedoiksi.

4.2 Vaatimusmäärittely

Vaatimusmäärittely on ohjelmistokehityksen vaihe, jonka aikana pyritään kokoamaan kaikki ohjelmistoa sekä mahdollisia ulkoisia komponentteja koskevat vaatimukset. Vaatimukset ovat erityisesti monimutkaisissa järjestelmissä lähtöisin useista eri lähteistä [63, s. 286-287]. Yleistetysti vaatimusmäärittelyvaiheen tuotoksena syntyy dokumentoitua tietoa järjestelmän odotetusta toiminnallisuudesta, mikä toimii suunnittelun lähtökohtana.

Vaatimustenhallinta liittyy hyvin tiukasti vaatimusmäärittelyyn. Vaatimustenhallinnalla tarkoitetaan prosessia, jonka avulla seurataan vaatimusten tiloja ja tuodaan muutoksia jo hyväksyttyyn vaatimusmäärittelyyn. Vaatimuksiin liittyvät tehtävät on esitelty kuvassa 4.1.



Kuva 4.1: Vaatimuksiin liittyvät aiheet [34].

Vaatimus on määritelty IEEE:n ohjelmistotuotannon sanastossa [35] suomennettuna seuraavasti:

1. Edellytys tai kyky, joka ratkaisee ongelman tai saavuttaa tavoitteen, jonka käyttäjä tarvitsee.

2. Edellytys tai kyky, jonka järjestelmän tai järjestelmän osan täytyy täyttää, jotta määritellyn sopimuksen, standardin tai jonkin muun virallisesti määritellyn dokumentin edellytys toteutuu.

3. Dokumentoitu esitys edellä mainituista.

Ohjelmistovaatimukselle voidaan esittää laatuattribuutteja, eli laatua kuvaavia ominaisuuksia. Wiegers esittää kirjassaan ”Software requirements” [34] vaatimukselle seuraavia laatuattribuutteja:

- **Täydellinen**: Vaatimuksessa täytyy olla täysin kuvattu haluttu toiminnallisuus. Siinä on oltava kaikki tarpeellinen tieto, jotta haluttu ominaisuus voidaan suunnitella ja toteuttaa.

- **Oikea**: Vaatimuksen täytyy kuvata tarkasti haluttu toiminnallisuus — oikeellisuus on osoitettavasti vaatimuksen lähteestä. Mikäli lähteen ja vaatimuksen välillä on ristiriita, on vaatimus väärä.

- **Toteuttamiskelpoinen**: Vaatimus tulee voida toteuttaa järjestelmän tai ympäristön rajoitteiden puitteissa.
- **Tarpeellinen**: Vaatimuksen tulee kuvata järjestelmän kyky, jonka käyttäjä tai ulkoinen rajapinta vaatii.
- **Priorisoitu**: Vaatimuksella tulee olla oma yksilöity prioriteetti, jonka mukaan vaatimusten toteuttamistarve voidaan todeta.
- **Yksiselitteinen**: Vaatimuksen tulee olla tulkittavissa aina samalla tavalla. Tämä on luonnollisella kielellä vaikeaa. Vaatimus tulee kirjoittaa yksinkertaisesti, käyttäen kohdealueen sanastoa.
- **Testattava**: Vaatimus tulee olla osoitettavissa oikein toteutetuksi (verifioitavissa) testeillä.

Koko vaatimusmäärittelydokumentaatiolle Wiegers esittää seuraavia arvoja:

- **Täydellinen**: Vaatimuksia tai tietoa ei saa puuttua.
- **Johdonmukainen**: Johdonmukaiset vaatimukset eivät ole ristiriidassa toistensa kanssa.
- **Muokattava**: Dokumenttia tulee voida muokata ja muutoshistoriaa on ylläpidettävä.
- **Jäljitettävä**: Vaatimukset tulee olla jäljitettävissä vaatimuksen lähteeseen, toteutukseen ja testaukseen.

Mike Mannion ja Barry Keepence esittävät laadukkaiden ohjelmistovaatimusten kuvaamiseksi SMART-muistisääntöä [71]. Muistisäännön mukaiset vaatimukset ovat täsmällisiä (*specific*), mitattavia (*measurable*), saavutettavissa olevia (*attainable*), toteutettavia (*realisable*) ja jäljitettäviä (*traceable*). Edellä esitetyt vaatimusten laatuattribuutit kuvaavat enemmän vaatimuksen esitystapaa kuin sen sisältöä. Esitetyistä laatuattribuuteista täydellisyys, tarpeellisuus ja oikeellisuus esittävät vaatimuksen sisällön laadukkuutta. Näiden todentamista voidaan pitää vaikeimpana — erityisesti täydellisyyttä on vaikea todistaa oikeaksi [34].

Vaatimuksia ja odotuksia järjestelmään voi kohdistua useilta eri osapuolilta (*engl. stakeholders*). Osapuolia ovat kaikki, joita kehitettävä järjestelmä jollain tavoin koskee. Tällaisia ovat esimerkiksi loppukäyttäjät eri rooleineen, ulkoiset järjestelmät, ylläpitäjät, yrityksen johto ja viranomaiset. Kehitettävän tuotteen vaatimuksiin voi

myös vaikuttaa eri säädökset ja lait. Erityisesti viimeksi mainitut vaikuttavat lääkinnällisten laitteiden ohjelmistokehitykseen.

Wiegersin kuvaamia vaatimusmäärittelyvaiheen tehtäviä on esitetty kuvassa 4.1. Vaiheet kohdistuvat vaatimusten löytämiseen, analysointiin ja esittämiseen. Vaiheet ovat lyhyesti esiteltynä seuraavanlaiset:

Vaatimusten esillesaaminen (*engl. elicitation*) on vaihe, jonka aikana eri lähteistä kootaan odotukset järjestelmää kohtaan. Tähän vaiheeseen on erilaisia tekniikoita, kuten haastattelut, työnkulun ja vastaavien järjestelmien dokumentaation tutkimista sekä järjestelmän kohderyhmän työtehtävien tutkiminen.

Analyysi on vaihe jonka aikana kerätyt vaatimukset arvioidaan tarkemmin. Vaatimukselle voidaan antaa esimerkiksi arvo, asettaa alustavia prioriteetteja ja etsiä mahdolliset riippuvuudet eri vaatimusten suhteen.

Neuvottelu käydään kaikkien niiden osapuolien kanssa, joita vaatimukset koskevat. Neuvottelulle on tarve kun vaatimuksille asetetaan tarkat prioriteetit ja selvitetään ongelmia vaatimuksissa. Erilaisia ongelmia voivat olla päällekkäisyydet tai ristiriidat. Neuvottelun tarkoituksena on saada yksimielisyys siitä, että vaatimukset ovat silloisella ymmärryksellä oikeita ja toteutuskelpoisia.

Dokumentointi on prosessi, joka määrittää yhteisen muodon vaatimusten ja vaatimuksiin liittyvien tietojen dokumentointiin.

Verifiointi ja validointi on vaatimusmäärittelyvaiheessa prosessi, joka varmistaa, että vaatimukset eivät ole ristiriidassa eri osapuolien kanssa tai keskenään. Validoinnilla tarkistetaan myös, että vaatimukset, joita koskee jokin standardi tai säännös, ovat oikein. Tällainen vaatimus voi olla esimerkiksi lääkinnällisessä laitteessa sykkeen mittaaminen. Tällöin validoinnilla varmistetaan, että vaatimuksessa esitellyt parametrit ja olettamukset ovat oikein lääketieteellisestä näkökulmasta.

4.3 Vaatimusten kirjaaminen

Usein vaatimukset kirjataan vaatimusmäärittelydokumentaatioon käyttäen luonnollista kieltä. Wiegers esittää, että vaatimukset voidaan dokumentoida jäsenneltynä luonnollisilla kielillä, graafisilla malleilla tai formaaleilla matemaattisesti tarkoilla loogisilla kirjoitustavoilla [34, l. 10]. Vaatimuksille hän esittää eri tasoja, kuten liiketoiminnan ja käyttäjien odotukset järjestelmää kohtaan sekä toiminnalliset ja laadulliset vaatimukset. Käyttäjien odotuksia voidaan kuvata käyttötapauksina (*use case*) tai käyttäjätarinoina (*user story*).

Käyttötapaukset ovat yleinen tapa ja de facto –standardi ohjelmistovaatimusten analysoinnissa ja määrittelyssä [44]. Käyttötapauksilla kuvataan toimijan (*ac-*

tor) ja järjestelmän (*system*) välinen interaktio. Tavoitteena on kuvata kuinka järjestelmän avulla saavutetaan tietty päämäärä tai tehdään tehtävä [34, l. 8]. Käyttötapauksien vahvuus on se, että niiden avulla voidaan kuvata järjestelmän toimintaa ilman, että tarkempaa suunnittelua on tehty. Käyttötapaukset eivät ota kantaa siihen kuinka järjestelmä tehdään, vaan ne kuvaavat käyttäjien odotuksia järjestelmää kohtaan. Käyttötapauksia voidaan myös kuvata graafisesti tietovuo- tai UML-aktiviteettikaavioilla. Wiegers esittää käyttötapauksien koostuvan seuraavista osista:

- **ID**: Yksikäsitteinen tunnistetieto.
- **Nimi**: Käyttötapauksen nimi muodossa verbi + objekti, esimerkiksi ”jättää tilaus”.
- **Kuvaus**: Lyhyt kuvaus käyttötapauksesta luonnollisella kielellä.
- **Ennakkoehdot**: Ehdot, joiden on oltava tosia ennen käyttötapausta.
- **Jälkiehdot**: Ehdot, joiden on oltava tosia käyttötapauksen jälkeen.
- **Reitti**: Numeroitu lista toiminnoista, joilla ennakkoehdoista päädytään jälkiehtoihin.
- **Poikkeavat reitit**: Normaalista reitistä poikkeava lista toiminnoista, joilla saavutetaan jälkiehdot.
- **Poikkeukset**: Toimintojen aikana tapahtuvat poikkeukset tai vikatilanteet ja niiden käsittely.

Käyttäjätarinat ovat ketterissä menetelmissä yleinen tapa kirjata käyttäjän odotuksia järjestelmää kohtaan [72, s. 3]. Käyttäjätarina koostuu toimijasta, toiminnosta ja kohteesta [72, s. 4]. Cohn laajentaa tätä esitystapaa vielä toiminnon hyödyllä. Käyttäjätarina kirjataan muodossa: *roolina* haluan, että *jotain*, jotta *hyöty*. Yksi käyttäjätarina voisi siis olla esimerkiksi: ”Opiskelijana haluan, että voin osallistua tenttiin Korpista, jotta minun ei tarvitse lähettää tenttikuorta”. Tarinan lisäksi kirjataan myös vaatimuksen koko ja prioriteetti.

Vaikka käyttötapaukset ja käyttäjätarinat kuvaavat järjestelmään kohdistuvia toiminnallisia odotuksia, ne eivät sellaisenaan ole riittävän tarkkoja kuvaamaan teknisiä toiminnallisia vaatimuksia [34, l. 8]. IEEE:n standardi 830 esittää toiminnallisten vaatimusten kirjaamiseksi seuraavaa muotoa: ”Järjestelmän pitää ... ” [77, s. 17]. Esimerkiksi suorituskykyvaatimus kirjattaisiin seuraavasti: ”95 % tapahtumista pitää

prosessoida alle 1 sekunnissa". Wiegers ehdottaa käyttämään toimijana tunnistettua käyttäjäroolia, kuten esimerkiksi: "Ostajan pitää hyväksyä tilaus erillisestä näkymästä". Näin kirjatuissa vaatimuksissa on selkeä toiminto, parametrit ja odotettava tulos. Näiden tietojen pohjalta vaatimukselle voidaan kirjoittaa kattava testi.

4.4 Lakien ja säännösten huomioiminen vaatimusmäärittelyssä

Vaatimusmäärittelijät, kehittäjät ja auditoijat kohtaavat kaksi haastetta lainmukaisuuden täyttämisessä ja tarkistamisessa: järjestelmää koskevien säännösten tunnistaminen ja niiden täyttäminen. Myös muiden osapuolien – erityisesti järjestelmän tilaajien – tulee ymmärtää säännökset, jotka koskevat järjestelmää. Heidän tulee osata vastata kysymyksiin, mikä on sallittua ja mikä ei [31].

Lain tai säännöksen huomiointi ja sen täyttäminen (*engl. compliance*) etenee seuraavalla tavalla:

- Tunnista järjestelmää koskevat säännökset,
- tunnista vaatimukset järjestelmälle ja
- vastaa määriteltyihin kyselyihin lainmukaisuuden testaamiseksi.

Vaatimusmäärittelyn alussa järjestelmää koskevien lakien ja säännösten tunnistaminen on avainroolissa lääkinnällisen laitteen ohjelmistokehityksessä [32]. Vaatimusten tunnistamisessa nousee esiin usein jokin säännös, josta viitataan taas toiseen säännökseen. Näiden eri säännösten välillä voi olla käytössään aivan toinen sanasto ja käsitteistö. Tällöin tulkinta ja tarkempi analysointi tulevat isoksi osaa vaatimusmäärittelyä [31]. Eri lakien ja säännösten viittauksien löytämiseksi avuksi voi käyttää erilaisia dokumentteja, joissa viitteet on merkitty. Näin voidaan tutkia onko kaikki oleelliset viittaukset käyty läpi. Säännökset ja lakitekstit ovat useissa maissa laitettu kaikkien saataville Internet-palveluihin, joten pääsy lakiteksteihin on entistä helpompaa.

Lait ja säännökset ovat hyvin rakenteellisia ja hierarkkisia tekstejä. Niiden tulkinta on usein vaikeata. Tämän vuoksi useisiin säännöksiin on tehty opaskirjoja, siitä kuinka lakia tulee tulkita. Esimerkiksi USA:n Department of Health and Human Services julkaisee ohjeistuksia kuinka otetaan käyttöön HIPAA Security Rule (Health Insurance Portability and Accountability Act) -säännös. Suomessa hieman vastaavaa työtä tekee VTT (Valtion teknillinen tutkimuskeskus), joka on tehnyt useita julkaisuja lääkintälaitedirektiivin vaikutuksesta lääkintälaitteiden kehitykseen, niin ohjelmistokehityksen kuin koko organisaation kannalta.

Säännösten ymmärtämiseen on esitetty monia eri tapoja [31], [32], [33], jotka ovat periaatteeltaan samoja kuin ohjelmistoa koskevien vaatimusten esittäminen muussa kuin luonnollisissa kielissä. Yksi tapa on muuttaa esitys formaaliin muotoon. Formaali muoto vaatii lukijaltaan ymmärryksen esitystavasta ja ei ole näin sopiva kaikille osapuolille. Samoin tutkimuksissa on havaittu, että formaalilla tavalla esittäminen johtaa joskus yksikäsitteisesti vääriin tulkintoihin [31].

Yksi tapa on purkaa säännös, tulkita se ja kirjoittaa siitä taulukko viittauksineen. Travis D. Breaux ja kumppanit esittävät seuraavanlaisen esimerkin tutkimuksessaan [33] ja sama purettuna ehdotetulla tavalla on esitelty taulukossa 4.1:

> Säännös §1194.21(d): Kun kuva esittää ohjelman osaa, esitetty tieto täytyy myös olla saatavilla tekstinä.
>
> Sana ”täytyy” (*engl. must*) tarkoittaa sitä, että vaatimus on pakollinen. Toimintolause ”olla saatavilla” tarkoittaa sitä toimintoa, joka täytyy tehdä, eli ”tehdä saataville”.

Teksti	*Ominaisuus*	*Arvo*
1194.21(d)	Edellytys	*Kun*... kuva esittää ohjelman osaa
1194.21(d)	Kohde	Tieto, jota kuva esittää
1194.21(d)	Toiminto	Tehdä saataville
1194.21(d)	Tapa	Teksti

Taulukko 4.1: Esimerkki säännöksen purkamisesta.

Lainmukaisuuden tarkistamisen voi tehdä erilaisilla tarkistuslistoilla, joita lainmukaisuuden auditoijat käyttävät. Lääkinnällisten laitteiden osalta tarkistuksen tekee niin sanottu ilmoitettu laitos (*notified body*), jollainen on suomessa VTT [10]. Listoissa on esitetty säännösten vaatimat kohdat, jotka tulee osoittaa täytetyiksi dokumentaatiolla. Viittaukset tulevat siis järjestelmän suunnittelu-, vaatimusmäärittely- ja testausdokumentaatioon. Viittaukset voi tehdä molempiin suuntiin, mikä osaltaan parantaa jäljitettävyyttä ja ymmärrettävyyttä [31]. Tällöin jokaisessa vaatimuksessa on viite suunnitteluun ja testaukseen sekä jokaisessa suunnittelu- ja testausdokumentista on viite vaatimuksiin. Kevyempi tapa on merkitä viitteet suunnittelusta vaatimuksiin vain järjestelmän kannalta kriittisimmistä osista. Viitteiden ylläpitämisen vuoksi jäljitettävyyden ja vaatimustenhallinnan rooli järjestelmän elinkaaren aikana on merkittävä.

Lääkinnällisten laitteiden vaatimusmäärittelyssä otetaan suunnittelun lähtötietovaatimuksiksi aiotun markkina-alueen säännökset. Tämän vaatimus tulee

ISO 13485:sta. Mikäli järjestelmä aiotaan tuoda usealle markkina-alueelle, tulee ottaa huomioon siis kaikkien alueiden viranomaisvaatimukset. Tämä entisestään lisää säännösten tulkitsemista ja analysointia vaatimusmäärittelyvaiheessa [16], [15, s.9-16]. Lääkinnällisiä laitteita koskee EU:n alueella lääkinnällisten laitteiden direktiivi ja joukko standardeja, joihin säännös löyhästi viittaa. Lääkinnällisten laitteiden direktiivi pyrkii näin ollen harmonisoimaan käytänteet korkealla tasolla koko Euroopan alueella. Direktiivin asettamista vaatimuksista on kerrottu tarkemmin kappaleessa 5.

Pöyhönen ja Hukki esittävät raportissa *"Riskitietoisen ohjelmiston vaatimusmäärittelyprosessin kehittäminen"*, että hyvä vaatimusmäärittely edellyttää määritellyn vaatimusmäärittelyprosessin lisäksi riskienhallinnan ja suunnittelukatselmoinnit, joita tehdään jo vaatimusmäärittelyn aikana [11, s. 9]. Tunnistetuille vaatimuksille tehdään riskianalyysi ja vaatimukset voidaan määritellä kriittisyyden mukaan. He korostavat vaatimusmäärittelyn aikaisen riskianalyysin vaativan kokemusperäistä tietoa järjestelmästä ja sen ympäristöstä. Tällöin erilaisten asiantuntijoiden käyttö jo riskianalyysin varhaisessa vaiheessa on suositeltavaa.

Erityispiirteenä lääkinnällisten laitteiden suunnittelussa on riskienhallinnan, verifiointi- ja validointisuunnitelmien sekä alustavan riskianalyysin aloittaminen jo vaatimusmäärittelyvaiheessa [15, s. 9], [11, s. 15]. Lääkinnällisen ohjelmistoa sisältävän järjestelmän arviointi tapahtuu sen vaatimusmäärittely-, suunnittelu- ja testausdokumenteista [15]. Erityisesti jäljitettävyys vaatimuksesta lähteeseen, toteutukseen ja verifiointiin on osoitettava järjestelmän arvioinnissa [15, s. 24], [16].

Lait ja säännökset eivät siis vaikuta pelkästään ohjelmiston ominaisuuksiin vaan koko tuotteen suunnitteluun, kehitykseen ja kehitystä tukeviin prosesseihin.

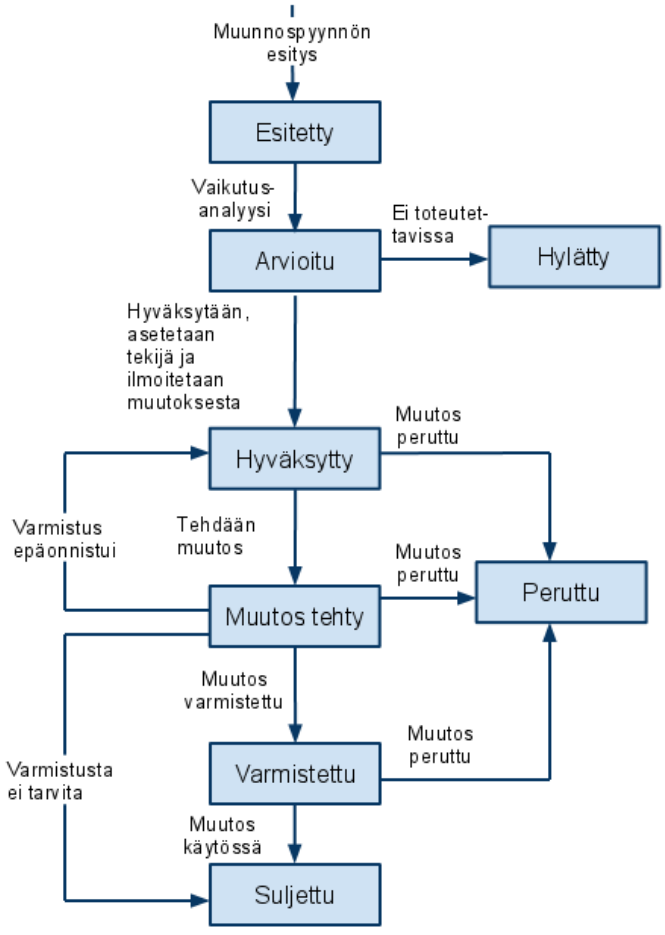
4.5 Vaatimustenhallinnan keskeiset toimet

Vaatimustenhallinta on prosessi, jonka avulla seurataan vaatimuksia ja tuodaan hallitusti muutoksia ohjelmistokehitykseen aktiivisen kehityksen tai ylläpidon aikana. Ymmärrys tehtävästä ohjelmistosta, ympäristöstä ja rajoitteista tarkentuu ohjelmistokehityksen aikana, eivätkä alussa tunnistetut vaatimukset ole aina oikeita tai täydellisiä. Vaatimukset muuttuvat, vanhoja poistuu ja uusia vaatimuksia ilmestyy kesken ohjelmistokehityksen ja ne pitää tuoda ohjelmistokehitykseen hallitulla tavalla. Muutos on luonnollista, eikä sitä pidetä huonona asiana [34, s. 328], [73].

Vaatimustenhallinnan keskeiset toimet on esitetty kuvassa 4.1 ja ne ovat: muutostenhallinta, versiohallinta, tilahallinta ja jäljitettävyys. Tärkein toimi projektin aikana on määritellä itse vaatimustenhallintaprosessi. Vaatimustenhallinnan voi näh-

dä kattotoimena [63, s. 6], johon kuuluu vaatimusten tilojen ja muutosten seuranta sekä eri tuoteperheiden hallinta. Vaatimustenhallinnan tulisi kuvata kuinka eri vaatimustenhallinnan tehtävät toteutetaan ja mitä työkaluja tehtävien kanssa voidaan käyttää.

Muutosten tuominen ei ole pelkästään dokumenttien hallintaa, vaan sen vaikutus kattaa koko järjestelmän. Uusi vaatimus tai muutos käsitellään samoin kuin alkuperäiset vaatimukset ja näiden tehtävien lisäksi otetaan huomioon nykyiset vaatimukset sekä toteutus. Ehdotetut muutokset voidaan arvioida nykyistä toteutusta vasten, eli arvioidaan suunnitellun arkkitehtuurin ehdoin, kuinka vaatimus voidaan toteuttaa ja mikä sen hinta on. Muutostenhallinta kuvaa kuinka uudet ja muuttuneet vaatimukset tuodaan projektiin. Wiegers esittää muutostenhallinnalle kuvan 4.2 kaltaista prosessia.



Kuva 4.2: Vaatimuksen muutostenhallinta [34, s. 355].

Muunnospyyntöjä tulisi esittää yhtenäistä kanavaa pitkin, jotta niiden vaikutus voidaan arvioida ennen kuin muutokselle tehdään muita toimenpiteitä. Tällä ta-

valla pyyntöjä saadaan karsittua ja niiden vaikutuksesta tiedetään enemmän ennen kuin varsinaisesta muutosta toteutetaan [74, s. 82]. Muutostenhallinnan kautta esitettävät muutospyynnöt voivat tulla useilta eri rooleilta, kuten yksittäisiltä käyttäjiltä, markkinoinnista tai muulta tuotteeseen liittyvältä osapuolelta. Arvioinnissa otetaan huomioon muutospyynnön vaikutuksen laajuus ohjelmistoon, kustannukset, hyödyllisyys ja riskit. Arvioinnin jälkeen muutospyyntö joko hylätään tai hyväksytään. Hyväksymisperusteina voidaan käyttää uuden vaatimuksen toteutuksen kustannus-hyötysuhteen laskemista.

Jotkin vaatimukset voivat olla kustannuksiltaan ja vaikutuksiltaan niin merkityksettömiä ettei niitä tarvitse erikseen varmistaa. Nämä muutokset voidaan toteuttaa Wiegersin mukaan kevyemmin ja ohittaa varmistusvaihe kokonaan. Varmentamisessa tarkistetaan, että muutos on viety ohjelmistoon siten, kuin se oli alun perin esitetty muunnospyynnössä. Samoin varmistutaan siitä, ettei vaatimus ole ristiriidassa ohjelmiston muuhun toiminnallisuuteen tai rajoitteisiin nähden.

Versiohallinnan tehtävänä on ylläpitää vaatimusten ja ohjelmistoversioiden versionumeroita. Tarkoituksena on estää eri versioiden ja vanhentuneen tiedon aiheuttamat ongelmat sekä mahdollistaa jäljitettävyys. Versiohallinta on käytännössä versionumerointia yksilöllisellä tunnisteella ja muutoshistorian ylläpitoa. Muutoshistoriasta täytyy Wiegersin [34, s. 317-319] mukaan käydä ilmi, mitä on muutettu, milloin on muutettu, kuka on muuttanut ja miksi on muuttanut. Versiohallinnan tehtäviin voidaan myös nähdä vaatimusten ja muiden versiohallinnan alaisuuteen kuuluvien dokumenttien jakelu määritellystä paikasta oikeille henkilöille.

Tilahallinnalla tarkoitetaan vaatimusten tilan kuvaamista. Wiegers ehdottaa yksittäisen vaatimuksen tilan kuvaamiseksi joukkoa valmiiksi määriteltyjä tiloja valmiusprosenttien sijaan. Valmiusasteen kuvaaminen prosentuaalisesti on hankalaa ja ne usein johtavat vääriin arvioihin. Vaatimusten tiloiksi voidaan esittää esimerkiksi *Ehdotettu*, *Poistettu*, *Ei aloitettu*, *Aloitettu*, *Toteutettu* ja *Valmis*. Uusille tai muuttuneille vaatimuksille asetaan tilaksi *Ehdotettu*, joka analyysin jälkeen asetetaan *Ei aloitettu* tilaan. Vaatimuksen tila asetetaan *Aloitetuksi*, kun kyseistä vaatimusta toteutetaan. Toteutettu vaatimus voidaan merkitään valmiiksi, kun kaikki osapuolet toteavat vaatimuksen valmiiksi. Vaatimuksen *Valmis*-tilan edellytyksenä voidaan myös pitää vaatimukselle suoritettavien verifiointi- ja validointitehtävien läpäisyä. Kaikkia vaatimuksia ei välttämättä aina toteuteta. Wiegersin mukaan poistetut vaatimukset kannattaa kuitenkin säilyttää jatkoa varten ja poistetuista vaatimuksista kirjataan hylkäämisen syy [34, s. 323]. Tilojen muutokset kirjataan ylös versionhallinnan määrittelemällä tavalla, jotta jäljitettävyys säilyy ohjelmiston eri versioihin.

Jäljitettävyys esitettiin aikaisemmin hyvän ohjelmistovaatimuksen laatuattribuu-

tiksi. Jäljitettävyyden ylläpito kuvataan usein vaatimustenhallinnan tehtävänä ja tarkoitus on ylläpitää vaatimuksista viitteet niiden lähteisiin ja suunnitteludokumentaatioon [71], [34, s. 354]. Tämän avulla muutosten vaikuttavuuden arvioinnissa voidaan käyttää hyväksi jäljitettävyyden tarjoamia linkkejä eri suunnittelu- ja määrittelydokumentteihin. Muutospyyntöjen yhteydessä tutkitaan viitteiden avulla vaikuttaako kyseinen vaatimus mahdollisesti toisiin vaatimuksiin ja mihin tuotteen osiin muutokset kohdistuvat toteutuksessa.

4.6 Vaatimusmäärittely ja vaatimustenhallinta eri malleissa

Luvussa 2 jaettiin ohjelmistokehitysmallit kahteen luokkaan: perinteisiin ja ketteriin menetelmiin. Perinteiset mallit ovat luonteeltaan suunnitelma- ja dokumenttivetoisia kun ketterät menetelmät puolestaan korostavat ihmisten kommunikaatiota. Vaatimusmäärittely ja vaatimustenhallinta ovat puolestaan osa perinteistä ohjelmistotuotantoa. Sen metodeihin kuuluu vaatimusten tunnistamista, analysointia, dokumentointia ja validointia, mistä johtuen sitä pidetään raskaana prosessina. Ketterät menetelmät nähdään usein epäyhteensopivina edellä kuvatun raskaahkon vaatimusmäärittelyn ja vaatimustenhallinnan kanssa [75].

Aikaisemmin esitellyssä V-mallissa on suoraan kuvattu mallin ensimmäiseksi vaiheeksi tutkimis- ja selvitystyö, jota voidaan pitää samankaltaisena kuin tässä luvussa esiteltyä esitutkimusta. Toisena vaiheena on vaatimusten kehittäminen, jossa voidaan käyttää suoraan esiteltyjä vaatimusmäärittelyn työkaluja. V-malli ei ota kantaa muuttuviin vaatimuksiin, joten sen rinnalle on esiteltävä vaatimustenhallinta omana prosessina tai muulla tavoin varauduttava muutoksiin.

Spiraalimallissa iteraatiot alkavat vaatimusmäärittelyllä. Malli ei ota kantaa kuinka tarkasti tai kuinka kauan vaatimusmäärittelyä tehdään. Spiraalimallin kehottama prototyyppien teko ja riskienhallinta iteraatioiden eri vaiheissa auttavat ymmärtämään ohjelmistosta enemmän, joten ensimmäinen iteraatio voidaan sellaisenaan käyttää kokonaan vaatimusmäärittelyyn. Muutoksiin spiraalimallissa on varauduttu sen iteratiivisella luonteella — vaatimusmäärittelyä tehdään aina iteraatioiden alussa. Varsinaista tapaa kirjata ja etsiä vaatimuksia ei malli esitä, joten ei ole mitään estettä tehdä vaatimusmäärittelyä ja vaatimustenhallintaa tässä luvussa kuvatuin metodein.

Ketteristä menetelmistä Scrum esiteltiin luvussa 3. Siinä vaatimusmäärittelydokumentaatio on tuotteen työlista, joka kehitetään alustusvaiheessa. Scrum ei määrittele millä tavoin ensimmäinen versio työlistasta kehitetään, joten sen apuna voidaan hyvin käyttää perinteisiä vaatimusmäärittelyn tehtäviä. Tuotteen työlista on

kuitenkin luonteeltaan erilainen kuin tässä kuvatun vaatimusmäärittelyn ja vaatimustenhallinnan vaatimusmäärittelydokumentaatio. Tuotteen työlistaa voidaan pitää vaillinaisena tai elävänä, koska siinä olevia vaatimuksia voidaan muokata, lisätä ja poistaa [75]. Vaatimustenhallinta on osa Scrumia; vaatimuksia voidaan muokata täysin vapaasti tuotteen työlistasta. Ainoastaan pyrähdyksen työlista on ”lukittu” muutoksilta. Pyrähdyksen katselmointipalaveria voidaan verrata vaatimusten esillesaamiseen, koska siinä esitetään kehiteteillä olevasta ohjelmistosta uudet toiminnallisuudet ja kerätään käyttäjiltä uusia vaatimuksia.

Corey Ladas esittää Lean-ajattelutavan mukaisessa Scrumban-mallissa vaatimusten kuvaamista käyttötapauksina (*use case*) [76, s. 27]. Scrumban ja kanban-järjestelmät eroavat vaatimusmäärittelyn esityön osalta muista malleista. Siinä vaatimuksia kerätään vain niin paljon kuin on tarpeen ja mallit ovat luonteeltaan imuohjattuja järjestelmiä [1] (*pull system*) [30, s. viii]. Täydellistä vaatimusdokumentaatiota ei yritetä saavuttaa. Yhdeksi periaatteeksi Ladas esittääkin: Älä tee enemmän määrittelyjä kuin voit ohjelmoida. Tärkein ominaisuus on tuottaa ohjelmiston tilaajalle mahdollisimman nopeasti arvoa [76, s. 27]. Nämä periaatteet eivät kuitenkaan sulje pois kaikkea vaatimusmäärittelystä opittua. Esillesaamistekniikat, varmistukset ja tarpeen tullen jäljitettävyyden ylläpito ovat vielä mahdollisia näissä malleissa.

Ketterät menetelmät eivät estä vaatimusmäärittelyn ja vaatimustenhallinnan vaiheiden käyttöä. Niitä voidaan käyttää mallikohtaisesti soveltuvin osin. Vaatimusmäärittelyä tai vaatimustenhallintaa ei välttämättä nähdä omana prosessinaan ketterien ohjelmistokehitysmallien rinnalla, vaan niiden toimintoja on itse mallin vaiheiden ja tehtävien sisällä [75].

4.7 Yhteenveto

Vaatimusmäärittelyn aikana tunnistetaan ohjelmistoa koskevat vaatimukset ja rajoitteet. Vaatimustenhallinnan menetelmillä ohjataan hallitusti muutosten tuonti sekä pidetään yllä laadukasta vaatimusmäärittelydokumentaatiota. Vaatimusmäärittelyvaihe on erityisen kriittinen lääkinnällisten laitteiden ohjelmistokehityksessä, koska se on vaihe, jossa tunnistetaan laitetta koskevat lait ja säännökset. Säännökset asettavat rajoitteita ja vaatimuksia ohjelmistokehitykselle, suunnittelulle ja kehitystä tukeville prosesseille. Säännösten tunnistamista voidaan pitää vaatimusmäärittelyn tärkeimpänä toimena lääkinnällisen laitteen onnistuneen ohjelmistokehityksen kannalta.

[1] Imuohjattu tarkoittaa, sitä että, tehdään mitä asiakas tarvitsee silloin ja vain silloin kun hän sitä tarvitsee.

5 Lääkinnällisen laitteen ohjelmistokehityksen vaatimukset

Tässä kappaleessa määritellään lääkinnällinen laite, esitellään siihen liittyvää lainsäädäntöä sekä tunnistetaan laitetta ja sen koskevat säännökset ja olennaisimmat harmonisoidut standardit. Näistä esitellään tarkemmin lääkinnällisen laitteen ohjelmistokehitystä koskevia vaatimuksia.

Lääkinnällinen laite on virallisesti määritelty eri maiden lainsäädäntöön. Euroopan parlamentti ja neuvosto määrittelee sen seuraavasti direktiivin 93/42/ETY [1] artiklassa yksi kohdassa kaksi:

> Tässä direktiivissä tarkoitetaan lääkinnällisellä laitteella kaikkia instrumentteja, laitteistoja, välineitä, materiaaleja tai muita tarvikkeita, joita käytetään joko yksinään tai yhdistelminä, sekä niiden asianmukaiseen toimintaan tarvittavia ohjelmistoja, joita valmistaja on tarkoittanut käytettäviksi ihmisten:
>
> - sairauden diagnosointiin, ehkäisyyn, tarkkailuun, hoitoon tai lievitykseen,
> - vamman tai vajavuuden diagnosointiin, tarkkailuun, hoitoon, lievitykseen tai kompensointiin,
> - anatomian tai fysiologisen toiminnon tutkimiseen, korvaamiseen tai muunteluun,
> - hedelmöitymisen säätelyyn.

Yhdysvaltojen FDA (*Food and Drug Administration*) määrittelee lääkinnällisen laitteen seuraavasti [21, s. 7]:

> Laite, instrumentti, kone tai jokin muu työkalu, jota käytetään vamman tai sairauden ehkäisyyn, diagnosointiin, hoitoon tai näiden toimenpiteiden apuna. Teollisuudessa termillä ”lääkinnällinen laite” tarkoitetaan tuotetta, jonka on FDA tai muu viranomainen tunnistanut laitteeksi lääkinnälliseen tarkoitukseen.

Lääkinnälliseksi laitteeksi voidaan siis ymmärtää mikä tahansa laite, ohjelmisto tai ohjelmistoa sisältävä laite, jonka avulla ihmisen terveydentilaa valvotaan tai

parannetaan. Tämän tutkielman kannalta olennaisinta on, että myös pelkkä ohjelmisto voidaan käsittää lääkinnälliseksi laitteeksi. Lääkinnälliseksi laitteeksi luettavia ohjelmistoja ovat esimerkiksi potilaan terveydentilaa valvovat ohjelmistot, hoidon suunnitteluun ja lääkeannosten koon määrittelyyn suunnitellut ohjelmistot sekä sähköiset potilas- ja lääketietokannat [21, s. 6].

5.1 Lääkinnällisten laitteiden direktiivi

Lääkinnällisen laitteen myyntiä ja kehitystä valvotaan sen markkina-alueen viranomaisten toimesta, jossa kyseistä tuotetta aiotaan myydä. ETA-sopimuksen määrittelemällä Euroopan talousalueella on voimassa EU:n säännökset, Yhdysvalloissa maan terveysministeriön (FDA) ohjeet ja Kiinassa myytävät tuotteet tarvitsevat CCC-sertifioinnin ja maan viranomaisten hyväksynnän [16], [21, s. 9-11], [15, s. 12]. Tässä tutkielmassa keskitytään terveydenhuollon laitteita ja tarvikkeita koskevaan EU-direktiiviin lääkinnällisistä laitteista (*Medical devices directive*) [1]. Direktiivin säännösten ja määräysten mukaan terveydenhuollon laitteet on suunniteltava ja valmistettava siten, että ne eivät vaaranna potilaan terveyden tilaa eivätkä vaaranna yhdenkään osapuolen turvallisuutta. Direktiivi koskee myös laitteissa olevan ohjelmiston suunnittelua ja toteuttamista [10].

Euroopan unionin direktiivi koskien lääkinnällisiä laitteita on alun perin säädetty kesäkuussa 1993 [1], jonka jälkeen sitä on muokattu viimeksi vuonna 1997 [2]. Tämän muutoksen seurauksena lääkinnällisen laitteen käsitys kattaa laajemmin ohjelmistot. Direktiivissä esitettiin huomiotavaksi ohjelmistojen kohdalta mm. seuraavia asioita:

”(6) On tarpeen selventää, että ohjelmisto yksinään on lääkinnällinen laite, jos ohjelmiston valmistaja on tarkoittanut sen käytettäväksi nimenomaan yhteen tai useampaan lääkinnällisen laitteen määritelmässä esitettyyn lääketieteelliseen tarkoitukseen. Yleisiin tarkoituksiin tarkoitettu ohjelmisto ei ole lääkinnällinen laite, kun sitä käytetään terveydenhoidon alalla.”

” (20) Koska ohjelmistojen merkitys kasvaa koko ajan lääkinnällisten laitteiden alalla sekä itsenäisinä yksiköinä että osana laitteita, olennaisena vaatimuksena olisi oltava parhaiden käytäntöjen mukainen ohjelmiston validointi.”

Näiden perustelujen myötä lisättiin tai muokattiin ohjelmistokehityksen kannalta seuraavat olennaiset asiat:

Luokittelusääntöihin (direktiivin liite IX) lisättiin määritelmä, jossa pelkkä ohjelmisto luokitellaan aktiiviseksi laitteeksi: ”Itsenäistä ohjelmistoa pidetään aktiivisena lääkinnällisenä laitteena.” Tämä merkintä nostaa automaattisesti ohjelmistoa

sisältävän laitteen riskiluokkaa ja tulee siten ottaa huomioon suunnittelussa.

Oleellisesti muuttuva vaatimus on direktiivissä liitteessä I, kohta 12.1a: ”Kun on kyse laitteista, jotka sisältävät ohjelmiston tai jotka ovat erillisiä lääketieteellisiä ohjelmistoja, ohjelmisto on validoitava parhaiden käytäntöjen mukaisesti ottaen huomioon kehityskaareen, riskinhallintaan, validointiin ja tarkastuksiin liittyvät periaatteet.”

Käytännössä direktiivi vaatii muutoksien jälkeen, että ohjelmistoa sisältävän lääkinnällisen laitteen tai lääkinnälliseksi laitteeksi luettavan ohjelmiston kehityksessä otetaan tarkemmin huomioon direktiivin asettamia vaatimuksia. Direktiivin vaatimukset täyttyvät, jos ohjelmiston kehitys noudattaa määriteltyä prosessimallia, jossa syntyy tarkistuksen vaatima dokumentaatio. Erityisesti suunnittelu, testaus (verifiointi), validointi ja riskienhallinta tulee olla dokumentoituna järjestelmällisesti. Ohjelmisto arvioidaan direktiivin ja standardien ISO 13485, ISO 14971 sekä IEC60601-1-4 vaatimusten pohjalta [10], [16].

Ohjelmistokehitysprosessin ei kuitenkaan tarvitse toteuttaa täysin määriteltyjä standardeja, vaan sen täytyy noudatella niitä. Tämä antaa vapauden käyttää tapaukseen sopivaa prosessimallia. On myös huomattava, että kaikki viranomaisvaatimukset täyttävät toimenpiteet eivät välttämättä ole määritelty ohjelmistokehitysprosessissa, vaan niitä tukevissa prosesseissa. Tällaisia prosesseja voivat olla esimerkiksi erilaiset testaustoimet ja riskienhallinta, joita toteutetaan ohjelmistokehityksen aikana erillisessä prosessissa ja jotka on määritelty esimerkiksi laadunhallintajärjestelmässä. Ohjelmistokehitysprosessin valinnassa on kuitenkin otettava huomioon näiden toisten prosessien tarpeet.

Suomessa uudistetun direktiivin sisältö on laissa 629/2010 ”Laki terveydenhuollon laitteista ja tarvikkeista”, joka tuli voimaan 1.7.2010 ja se kumoaa vanhan direktiivin mukaisen lain 1505/1994. Lain noudattamista Suomessa valvoo Lääkelaitos.

5.2 Lääkinnällisen laitteen laiteluokat

Euroopan unionin direktiivit [1], [2] vaativat, että lääkinnällisille laitteille määritellään laiteluokka. Laitteet jaetaan neljään eri luokkaan niiden monimutkaisuuden, riskiluokan ja aiotun käyttötarkoituksen mukaisesti. Aiottu käyttötarkoitus vastaa kysymykseen: *Mitä laitteella tehdään?* Erityisesti laitteen väärinkäytön tai laitteen riski aiheuttaa vahinkoa potilaalle vaikuttavat laitteen luokitukseen. Yhdysvaltojen terveysministeriön säännöksissä laitteet jaetaan samoin perustein kolmeen eri luokkaan. Käytännössä korkean riskiluokan laitteet käyvät läpi tarkemmin määritellyn ja dokumentoidun suunnittelu- ja tuotantoprosessin kuin alemman riskiluokan lait-

teet [3].

Tuotteen laiteluokka määritellään laitteen kokonaisriskiluokan mukaan, eli millaista vahinkoa laite voi pahimmillaan potilaalle tehdä. Tämä puolestaan määrittelee pitkälti sen kuinka ”raskaan prosessin” laitteen tai ohjelmiston suunnittelu, toteutus ja testaus tarvitsevat. Tässä tutkielmassa käsitellään laiteluokkia siten kuin ne on esitelty lääkinnällisiä laitteita koskevassa direktiivissä. Laiteluokat koskettavat näin ollen vain EU -markkina-alueen tuotteita. Laiteluokat ovat muunnettavissa helposti muiden markkina-alueiden laiteluokiksi, koska laiteluokkien määrittelyperiaatteet ovat yhtäläiset.

Laiteluokat on määritelty lääkinnällisten laitteiden direktiivissä [1, liite IX]. Laiteluokkia on neljä: I, II a, II b, III ja ne on jaettu pienemmän riskiluokan laitteista korkeamman riskiluokan laitteisiin. Direktiivissä ei ole tarkemmin määritelty mikä on ohjelmistoa sisältävän laitteen laiteluokka. Kuitenkin uudistettuun direktiiviin lisättiin kohta liitteeksi IX, jossa ohjelmistoa sisältävät laitteet määritellään aktiivisiksi lääkinnällisiksi laitteiksi, mikä automaattisesti tarkoittaisi sitä, että laitetta koskettaisi liitteen IX säännöt 9, 10 ja 12 suurimmassa osassa laitteita [14]. Näiden johtopäätöksien pohjalta Ruotsin Lääkelaitoksen työryhmä esittää oppaassaan [14] kohdassa seitsemän seuraavia ohjeita ohjelmistoa sisältävän laitteen laiteluokan määrittämiseksi:

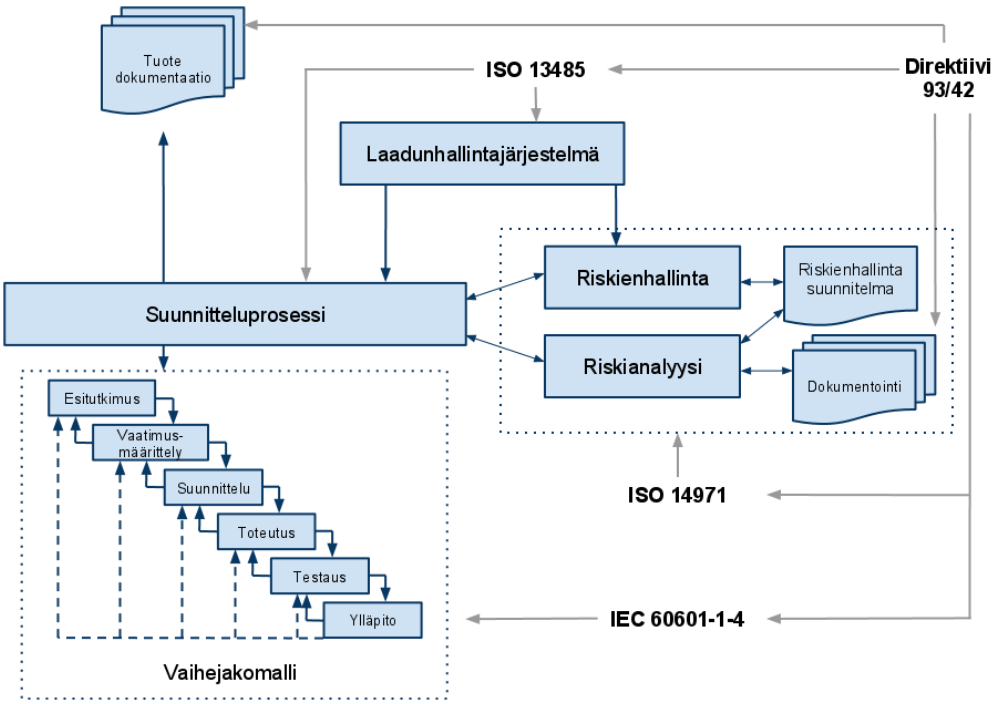
- Ohjelmisto, jonka tarkoitus on ohjata laitetta, joka ohjaa sähköä tai nesteitä on luokka II a tai II b, mikäli laitteella voi saada potilasvahingon aikaiseksi.
- Ohjelmisto, jonka tarkoitus on ohjata ja/tai monitoroida aktiivisen laitteen toimintaa on luokka II b.
- Ohjelmisto, joka on tarkoitettu diagnoosin tekoon on luokka II a.
- Ohjelmisto, joka on tarkoitettu erityisesti elintoimintojen tarkkailuun kuuluu luokkaan II b.
- Ohjelmistot, jotka eivät sovellu aikaisemmin esiteltyihin kuuluvat luokkaan I.

Laiteluokka ja aiottu käyttötarkoitus on määriteltävä heti suunnittelun alussa [15, s. 27]. Näiden tietojen pohjalta saadaan peruslinjaus tuotteen riskianalyysin, suunnittelun ja testauksen tarpeeseen. Kun peruslinjaus on selvä, voidaan valita tuotekehitykselle sopiva ohjelmistokehitysmalli.

5.3 Direktiivin asettamat vaatimukset

Lääkinnälliseen laitteeseen kohdistuu Euroopan unionin markkina-alueella yhtenäisesti direktiivi 93/42/ETY, josta esiteltiin aikaisemmin erityisesti ohjelmistoa koskevia kohtia. Säännöksen vaatimukset kohdistetaan koko tuotekehityksen prosessin kaikkiin niihin vaiheisiin, joilla tuote luodaan. Näin ollen valmistajan on määriteltävä ohjelmistokehitys yhtenä kokonaisuutena, jonka kyvykkyys tuottaa laadukasta ohjelmistoa arvioidaan säännöllisesti [10, s. 39]. Tämä vaatimus asettaa erityisiä vaatimuksia ohjelmistokehitysprosessia kohden, ja ottaen huomioon nykyisten ohjelmistojen koon sekä merkityksen nykyaikaisissa lääkinnällisissä laitteissa, soveltuvan ohjelmistokehitysprosessin valinta on yksi tärkeimmistä osista tuotteen kehityksen elinkaaressa.

Euroopan komissio on julkaissut Euroopan unionin virallisessa lehdessä listan direktiivin soveltamisalaan kuuluvista yhdenmukaistetuista standardeista [38]. Kuvassa 5.1 on esitetty direktiivin asettamat olennaisimmat vaatimukset ohjelmistoa sisältävän tuotteen kehityksen kannalta.



Kuva 5.1: Direktiivin viittaukset [11, s. 20].

Direktiivi ja sen soveltamiseen käytetyt standardit vaativat, että ohjelmistokehityksen tulee tapahtua määritellyn vaiheistetun ohjelmistokehitysmallin mukaisesti ja ohjelmistokehityksen eri vaiheissa tulee soveltaa riskienhallinnan työkaluja [11,

s. 5], [16, s. 86]. Ohjelmistokehitys ei näin ollen ole täysin erillinen suljettu prosessi, vaan sillä on jaettuja tehtäviä riskienhallinnan ja määritellyn laatujärjestelmän kanssa.

Säännöksen mukaisen suunnittelun ja toteutuksen osoittaminen ei tapahdu valmiista tuotteesta, vaan arvioimalla prosessia, jolla ohjelmisto on kehitetty [10, s. 11]. Arviointi kattaa koko ohjelmiston elinkaaren, joten siihen kuuluvat määrittely, suunnittelu, testaus, toteutus, verifiointi ja validointi sekä ylläpito. Prosessin arviointi tapahtuu ohjelmistojen osalta tutkimalla ohjelmistokehitysmallin tuottamaa dokumentaatiota – esimerkiksi vaatimusmäärittelystä tulee löytyä vaatimukset, joilla eliminoidaan tiettyjä riskejä ja suunnitteludokumentaation perusteella arvioidaan ohjelmiston vaatimustenmukaisuus [10, s. 40].

Olennaisten vaatimusten (*essential requirements*), eli direktiivin liitteen I vaatimukset liittyen suorituskykyyn ja turvallisuuteen, täyttyminen on osoitettavissa kun suunnittelu on tehty harmonisoituja standardeja noudattaen [15, s. 71]. Tämä on tarkistettavissa prosessien aikana tehdyistä dokumenteista. Näihin yksittäisiin dokumentteihin viitataan yleisesti *todisteina* ja kaikkiin dokumentteihin yhteisesti viitataan termillä *tekninen tiedosto* (*engl. technical file*) [21, s. 225], [15, s. 56]. Teknisen tiedoston yhteenveto -dokumenttilla (*engl. Summary technical document, STED*) tarkastetaan tuotteen direktiivin mukaisuus [21, s. 225], [39, s. 8-9]. Taulukossa 5.1 on esitelty teknisen tiedoston vähimmäissisältö ohjelmistoa sisältävän lääkinnällisen laitteen kohdalta.

Otsikko	*Alakohdat*	*Tarkennus*
Yleinen kuvaus tuotteesta	Tuotteen aiottu käyttötarkoitus, kohderyhmä, laiteluokka ja tuotteen kuvaus.	Esitutkimuksen ja projektin aloituksen aikana määriteltävät asiat. Voidaan sijoittaa esimerkiksi projektisuunnitelmaan.
Olennaisten vaatimusten täyttyminen	Soveltuvat vaatimukset	Esitutkimuksen ja vaatimusmäärittelyn aikana tunnistetut sitovat säännökset ja vaatimukset. Sijoitetaan vaatimusmäärittelydokumentaatioon.

	Vaatimusten täyteen osoittaminen	Tunnistetut vaatimukset ja niiden täyttyminen on oltava osoitettavissa tuotteen suunnitteludokumentaatiosta.
	Käytetyt standardit	Tunnistetut koskevat standardit listataan joko projektisuunnitelmaan tai vaatimusmäärittelydokumentaatioon.
Valmistusmenetelmät	Valmistusprosessin kuvaus	Mikäli tuotteeseen kuuluu fyysinen laite, on tämän valmistusprosessi kuvattava.
	Laadunvarmistuskäytänteiden kuvaus	Laadunvarmistuksen tehtävät, määritellyt laatuattribuutit ja valvontaprosessin kuvaus.
	Suunnittelu- ja kehitysprosessin kuvaus	Kuvaus tuotteen suunnittelussa ja toteuttamisessa käytetyistä prosesseista, esimerkiksi käytetty vaihejakomalli.
	Suunnitteludokumentaatio	Yhteenveto-dokumenttiin riittää korkean tason arkkitehtuurikuvaus tuotteesta.
Tekniset eritelmät	Elektroniikka, ohjelmistot, elinikä	Tuotteessa käytetyn elektroniikan ja kolmannen osapuolen ohjelmistojen eritelmät.
	Tuotteen kuvat, piirrustukset ja kaaviot	Fyysisen laitteen kuvaukset, ohjelmistoissa esimerkiksi eri topologioiden kuvaukset.
Suunnittelun todentaminen	Verifiointisuunnitelma	Yhteenveto kaikista tuotteelle suunnitelluista testeistä, esimerkiksi laboratorio-, ohjelmisto- ja rasitustestisuunnitelmat.
	Analyysien ja testien tulokset	Toteutettujen analyysien ja testien tuloksien yhteenveto.
Kirjallinen materiaali	Tuotteen merkinnät	Tuotteessa käytettyjen termien ja merkintöjen lista.

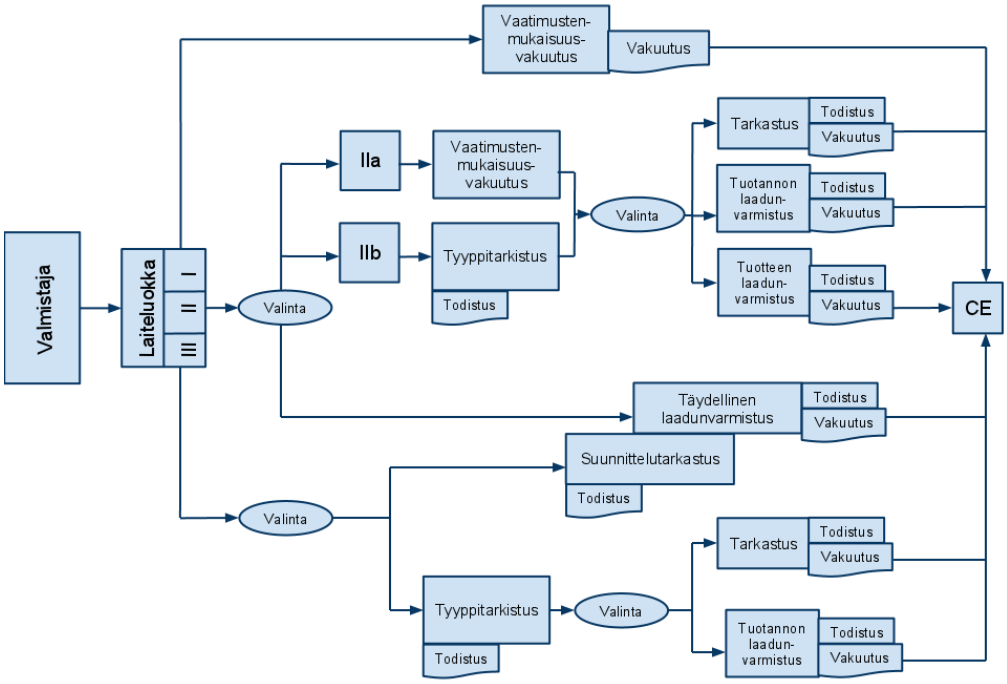
	Käyttö-, asennus- ja huolto-ohjeet	Jokaisen kielikäännöksen kirjallinen materiaali toimitetaan hyväksyntää varten.
Riskienhallinta	Riskienhallintasuunnitelma	Tuotekohtainen suunnitelma riskienhallintaan, jossa määritellään ainakin riskien arviointiprosessi ja hyväksyttävyystasot.
	Riskianalyysin ja seurannan dokumentaatio	Yhteenveto kaikista tunnistetuista riskeistä ja tehdyistä toimenpiteistä riskien pienentämiseksi hyväksyttävälle tasolle.
Kliininen osoitus	Kliinisen evaluoinnin suunnitelma	Tarvittaessa tehtävän kliinisen evaluoinnin suunnitelma.
	Kliininen evaluointiraportti	Kliinisen evaluoinnin tulokset.
Hallinnolliset dokumentit	Vaatimustenmukaisuusvakuutus	Liitetään osaksi teknistä tiedostoa kun olennaiset vaatimukset on osoitetusti täytetty.
	Aliurakkasopimukset	Tuotteen omistava yritys vastaa aliurakalla teetettyjen suunnittelun tai kehityksen tehtävien olennaisten vaatimusten täyttämisestä. Sopimuksessa todetaan aliurakoitsijan soveltuvuus ja pätevyys tuotteen kehitykseen.

Taulukko 5.1: Teknisen tiedoston sisältö [21, s. 225], [15, s. 91], [39, s. 8-15].

Tuotteelle vaaditaan CE-merkintä ennen kuin sitä voidaan myydä Euroopassa tuotteen suunniteltuun ympäristöön ja käyttötarkoitukseen. CE-merkintä on ikään kuin tuotteen passi, jonka avulla sitä voidaan myydä jokaisessa EU-maassa ilman

erillistä kansallista tarkistusta [21, s. 222]. Ilman CE-merkintää tuotetta voidaan käyttää tutkimuskäytössä, mikäli tuote ei olennaisesti vaaranna tutkittavan kohteen turvallisuutta. Tuote voidaan CE-merkitä täyttämällä direktiivin asettamat vaatimukset ja läpäisemällä ilmoitetun laitoksen hakuprosessin.

Laitteelle määritelty laiteluokka vaikuttaa hieman CE-merkinnän hankkimiseen ja tuotteen arviointiin. Kuvassa 5.2 on esitelty Suomessa sovellettavia hakureittejä. EU:n alueella arvioinnin toteuttaa ilmoitettu laitos (*notified body*). Suomessa kyseinen laitos on VTT Automaatio Terveydenhuollon tuotetekniikka, jonka valvovana viranomaisena toimii Lääkelaitos [16, s. 13]. Arviointi on aina tapauskohtainen ja arviointitapa riippuu ohjelmiston valmiusasteesta, kompleksisuudesta, kriittisyydestä ja yrityksen prosessimalleista [11, s. 23].



Kuva 5.2: Reitit tuotteen markkinoille saattamiseksi [11, s. 19].

Korkeamman riskiluokan laitteissa —laiteluokat IIa, IIb ja III— direktiivinmukaisuudenarvioinnin tekee ilmoitettu laitos. Tuotteen valmistajalla on muutamia eri vaihtoehtoja suorittaa arviointi. Tarkastaminen liittyy tuotteen suunnittelumenetelmiin, valmistukseen ja itse tuotteeseen. Tarkastaminen voidaan tehdä arvioimalla kehityksen aikana tehtyjä dokumentteja tai arvioimalla kattavammin yrityksen laatujärjestelmää ja dokumentaatiota [11, s. 20]. Valmistajan on annettava vakuutus direktiivin mukaisuudesta, joka on tarvittaessa voitava osoittaa todeksi. Arvion te-

kevä ilmoitettu laitos antaa todistuksen kyseisen tuotteen direktiivin täyttämisestä, jonka jälkeen tuote voidaan CE-merkitä.

Edellä esitetystä voidaan yhteenvetona esittää, että direktiivin vaatimukset täyttyvät kun:

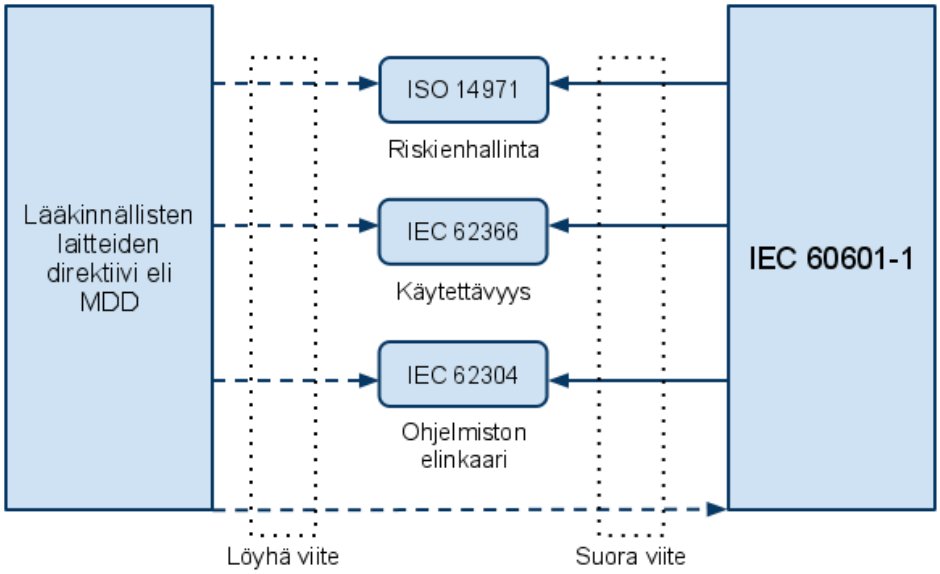
- Laiteluokka ja aiottu käyttötarkoitus on määritelty,
- tuotetta koskevat standardit ja säännökset on tunnistettu,
- olennaisten vaatimusten täyttyminen on todistettavissa,
- riskienhallintaprosessi on käytössä,
- suunnitteludokumentaatio on riittävää,
- tuotteen verifiointi- ja validointiprosessi on määritelty,
- ohjelmistokehitysmalli on määritelty ja
- ohjelmistokehitysmallilla on rajapinta riskienhallintaan.

5.4 IEC 60601

IEC 60601 lääkinnällisten elektronisten laitteiden standardi (*medical electrical equipment standard*) on International Electrotechnical Commission -komitean julkaisema ja ylläpitämä standardi. Se on esillä Euroopan komission julkaisemalla listalla yhdenmukaistetuista standardeista ja vastaa näin ollen direktiivin asettamiin vaatimuksiin. Tämän vuoksi IEC 60601 on yksi olennaisimmista standardeista, joita seuraamalla kehitetty tuote täyttää direktiivin vaatimukset ja jonka mukaan olennaisten vaatimusten täyttymisen arviointi on mahdollista suorittaa. Standardi ottaa kantaa kattavasti kaikkiin erilaisiin lääkinnällisiin laitteisiin kirurgisista laitteista mittaaviin ja ohjelmoitaviin laitteisiin. Tässä kappaleessa esitetään standardin olennaisimmat kohdat. Direktiivin ja standardin IEC 60601-1 viittaukset eri standardeihin on esitetty kuvassa 5.3. Standardi IEC 60601 koostuu neljästä osasta:

- 60601-1 perusvaatimukset (*general requirements*)
- 60601-1-x rinnakkaisstandardit (*collateral standards*)
- 60601-2-x erityisvaatimukset (*particular requirements*)
- 60601-3-x suorituskykyvaatimukset (*essential performance*)

Rinnakkaisstandardit koostuvat yleisistä standardeista erilaisille lääkinnällisille laitteille. Yleiset turvavaatimukset on määritelty erikseen lääkinnällisille laitteille, elektromagneettisille ja säteileville laitteille sekä erikseen omana kohtanaan ohjelmoitavat elektroniset lääkelaitteet. Erityisvaatimukset koostuvat 50:stä eri alakohdasta, jotka ottavat kantaa tarkemmin erilaisiin lääkintälaitteisiin, kuten säteileviin ja suoraan potilaaseen kosketuksessa oleviin laitteisiin. Suorituskykyvaatimukset on lisätty standardiin kolmannen version myötä ja ne koskevat laitteen olennaista suorituskykyä erilaisissa vaara- ja virhetilanteissa.



Kuva 5.3: Direktiivin ja IEC 60601-1 viitteet standardeihin [23].

IEC 60601 on kansainvälinen standardi ja useimmat maat ottavat käyttöön siitä harmonisoidun kansallisen version, johon on lisätty omaa markkina-aluetta koskevia pieniä muutoksia. Euroopan unionin alueella standardista on käytössä EN 60601. EU-alueen kansallinen standardi on täysin identtinen kansainvälisen standardin kanssa toisin kuin esimerkiksi USA, Kanada ja Japani ovat lisänneet omiin kansallisiin standardeihin maakohtaisia lisäyksiä.

Standardi määrittelee elektronisen lääkintälaitteen ja asettaa sille tiettyjä turvavaatimuksia [42]. Standardin on ns. ”vaara-spesifinen” (*hazard-specific*). Se asettaa vaatimuksia, joiden pohjalta voidaan arvioida yleisiä elektronisten lääkinnällisten laitteiden aiheuttamia vaaratilanteita. Useimmat näistä liittyvät sähkö- ja säteilyturvaan. Vaaroja tarkastellaan potilaan ja käyttäjien kannalta ja tarkoituksena on pienentää vaaratilanteen todennäköisyyttä [17]. Standardi siis antaa vaihtoehdon suunnitteluprosessin riittävyyden arvioinnille, jolloin suunnittelun aikana havaitut riskit

on pienennetty riskienhallintaprosessin avulla hyväksyttävälle tasolle [18].

IEC60601-1 perustuu idealtaan riskienhallintaan. Riskit määritellään ja hallitaan tuotteen suunnittelun, tuottamisen ja tuotteen aiotun käyttötarkoituksen pohjalta. Standardin 3. versioon on lisätty viite ISO 14971 riskienhallinnan osalta. Tämän myötä tuotteen riskienhallinnan lähtökohdaksi on otettava kyseinen standardi soveltuvin osin. Uudessa versiossa on lisätty myös jäännösriskien seuranta ja olennainen suorituskyky (*essential performance*). Riskienhallinta ei siis enää kata pelkästään laitteen suunnittelua vaan koko tuotteen elinkaaren.

Standardin esittämät vaatimukset ovat lähtökohtaisesti määritelty koskemaan laiteluokkia, joten tuotteella täytyy olla määritelty laiteluokka ennen kuin standardin asettamia vaatimuksia testataan. Standardin asettamien vaatimusten täyttymisen osoittaminen on käytännössä vaatimusten testausta — niiltä osin kuin se on mahdollista — ja tuotteen kehityksen aikana syntyneen dokumentaation katselmointia.

Ohjelmistokehityksen kannalta kiinnostavin ja tärkein on IEC 60601-1-4: Programmable Electrical Medical Systems [4]. Vaatimukset keskittyvät olennaisesti turvallisuuteen ja suorituskykyyn. Tämän myötä ohjelmiston suunnitteludokumentaatiosta tulee kyetä osoittamaan turvallisuus- ja suorituskykykriittiset osat. Ohjelmistoa tuleekin arvioida riskienhallinnan kautta löydettyihin riskeihin ja osoitettava, että vaaditut toimenpiteet riskien pienentämiseksi on otettu huomioon suunnittelussa. Standardi vaatii myös, että ohjelmistokehitysprosessi on dokumentoitu ja jokainen ohjelmistokehityksen vaihe on määritelty sekä jokaisella vaiheella on selkeät tulokset ja vaiheen esivaatimukset. Samoin riskienhallinnan tulee kattaa koko ohjelmistokehityksen elinkaari. Mitään valmista ratkaisua standardi ei esittele vaan määrittelee selkeät vaatimukset, mitä osoittavaa dokumentaatiota (*evidence*) ohjelmistokehityksen aikana tulee syntyä. Tämä antaa mahdollisuuden valita ja muokata olemassa olevia ohjelmistokehitysmalleja vastaamaan standardia.

Erityisesti huomiota kiinnitetään ohjelmiston arkkitehtuuriin. Standardi yksiselitteisesti vaatii, että ohjelmiston arkkitehtuuri on määritelty ja vastaa ohjelmistolle asetettuja vaatimuksia, niin laadullisia kuin toiminnallisia. Arkkitehtuurin vastaavuus voidaan osoittaa arvioimalla arkkitehtuuria laatupuun avulla tai verifioimalla kaikkien toiminnallisuuksien huomiointi arkkitehtuurista. Samoin ohjelmiston suunnittelusta, toteutuksesta ja testauksesta täytyy syntyä dokumentaatiota. Standardi ei sinällään ota kantaa, onko ohjelmiston arkkitehtuuri sama kuin suunnitteludokumentaatio, kunhan vaaditut kohdat voidaan osoittaa ja riskien kannalta kriittisimmät osat on testattu.

Standardissa vaaditaan myös verifiointi ja validointi. Verifiointi voidaan tehdä tuotetta kehittävän yrityksen sisällä, sillä sille ei ole asetettu riippumattomuusvaatimuksia. Verifiointi onkin usein helppo asettaa ohjelmistokehityksen prosessimallin eri vaiheiden väliin ikään kuin hyväksyntävaiheeksi ennen seuraavaa vaihetta [10]. Validoinnista standardi vaatii täyden riippumattomuuden kehittäjän ja validoivan elimen välille. Standardi ei vaadi kuitenkaan ohjelmiston validointia vaan koko tuotteen validointia, joten validointi useimmiten nähdään täysin erillisenä prosessina kehityksen ulkopuolella.

IEC 60601-1-4 lisäksi ohjelmistoja koskee standardi IEC 62304: Medical device software – Software life cycle processes, joka soveltuu myös yksittäisille ohjelmistoille, siinä missä 60601-1-4 koskee ohjelmistoja sisältävää elektronista laitetta [10, s. 11]. Standardia IEC 62304 ja ohjelmistokehitystä käsitellään tarkemmin luvussa 5.6.

5.5 Laadunhallinta

Laadunhallintajärjestelmät on suunniteltu ohjaamaan organisaatioiden toimintaa. Niissä toiminnot kuvataan prosesseina, jolloin toimintojen tavat ovat ohjeistettuja ja suunnitelmallisia. Tarkoituksena on parantaa laatua, tiedonkulkua ja vähentää inhimillisten virheiden tai ristiriitaisten tietojen ja käsitteiden vaikutteita suunnitteluun [16, s. 14]. Lääkinnällisten laitteiden suunnittelun ja valmistamisen tueksi on luotu oma standardi *ISO 13485 Medical devices — Quality management systems — Requirements for regulatory purposes*. Se on Euroopan unionin komission julkaisemalla listalla lääkinnällisten laitteiden direktiivin kanssa harmonisoiduista standardeista [38]. Laadunhallintajärjestelmän avulla ohjataan tuotteen suunnittelun ja kehityksen toimenpiteitä yhtäläisesti ja määritellään kaikki vaadittavat dokumentit, joita näistä prosesseista syntyy.

Standardi ISO 13485 on riskienhallintastandardin ISO 14971 ja standardin IEC 60601 kanssa yksi olennaisimmista standardeista, jotka tulee ottaa huomioon lääkinnällisen laitteen kehityksessä ja markkinoille saattamisessa. Monien maiden lääkinnällisten laitteiden hyväksymisjärjestelmä pohjautuu ISO 13485:een [15, s.12], joten standardin seuraamisesta on apua CE-merkintää haettaessa.

Lääkinnällisten laitteiden laatujärjestelmä on standardin ISO 9001 mukainen, mutta vaatii prosessien ja menettelyjen tarkempaa dokumentointia ja määrittelyä [15, s. 14]. Nämä vaatimukset koskevat puolestaan jokaista prosessia, johon kyseinen vaatimus kohdistuu. Esimerkiksi suunnitteludokumentaation tulee olla tarkempaa ja sille määritellään omat katselmointi- ja verifiointitoimenpiteet laatujärjestelmässä. Tämä näkyy esimerkiksi ohjelmistokehityksen vaihejakomallissa suun-

nittelun systemaattisena dokumentointina ja näiden katselmointeina.

Standardi vaatii mm. seuraavia toimenpiteitä, jotka voidaan nähdä koskevan myös ohjelmistokehitystä ja sen vaiheita [15, s. 14]:

- Määrätyt prosessit dokumentoidaan ja tallenteita ylläpidetään.
- Asetettujen vaatimusten täyttäminen ja mittaaminen.
- Korjaavien toimenpiteiden analysointi, suunnittelu sekä jäljitettävyys.
- Järjestelmän suorituskyvyn analysointi.
- Riskienhallinta koskee koko tuotteen elinkaarta.

Lääkinnällisen laitteen suunnittelussa ja kehityksessä laatua voidaan kuvailla seuraavanlaisesti [15, s. 12], [50, s. 3]:

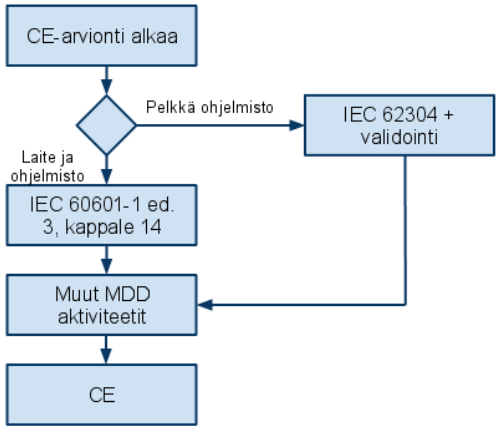
- Laatu tukee turvallisuutta ja suorituskykyä,
- turvallisuus ja suorituskyky tukevat kestävyyttä,
- kestävyys tukee joustavuutta,
- joustavuus tukee nopeutta,
- nopeus tukee kustannuksia.

Laadun kuvaukset eivät ole tasa-arvoisia siinä mielessä, että osa kuvauksista antaa enemmän arvoa tuotteelle kuin toinen. Esimerkiksi terveydenhuollon laitteen kannalta tärkein ominaisuus on turvallisuus ja suorituskyky, mutta käyttäjän kannalta tärkeämpää on käyttäjän odotukset tuotetta kohtaan. Laadunkäsite voidaan nähdä kahdella tavalla: (1) se kuvaa tuotteen ominaisuuksia täyttää odotukset ja (2) se kuvaa tuotteen ilman puutteita [50, s. 4]. Laadunhallinnan avulla pyritään tuottamaan jatkuvasti laadukasta tuotetta, jotta kaksi määriteltyä laadun käsitettä täyttyvät. Standardi ISO 13485 ohjaa suunnittelua ja edellyttää, että suunnittelun lähtökohdiksi otetaan kunkin markkina-alueen turvallisuus- ja suorituskykyvaatimukset ja elinkaaren aikainen riskienhallinta [15, s. 12].

Laatujärjestelmän ja direktiivin vaatimat elementit kuvataan tarkemmin riskienhallinnan, ohjelmistokehityksen ja verifiointi- sekä validointitoimenpiteiden osalta seuraavissa kappaleissa.

5.6 Ohjelmistokehitys

Lääkinnällisten laitteiden ohjelmistot ovat turvallisuuskriittisiä ja niiden todennäköisyys vikaantua täytyy olla pieni [24, s. 43]. Luvussa 5.3 todettiin, että lääkinnällisen laitteen ohjelmistokehitykseen liittyy vahvasti riskienhallinta, verifiointi- ja validointitoimenpiteet sekä näiden toimintojen systemaattinen dokumentointi. Direktiivin olennaisten vaatimusten täyttämisen arvioinnissa vaaditaan, että ohjelmistokehitysmalli on kuvattu ja sille on asetettu harmonisoiduissa standardeissa esitettyjä vaatimuksia, jotka kuvataan tässä kappaleessa. Ohjelmistojen yhdenmukaisuuden arviointi direktiivin kanssa suoritetaan soveltamalla, joko standardia IEC 60601-1-4 tai IEC 62304, kuten kuvassa 5.4 on esitetty.



Kuva 5.4: Ohjelmistoa sisältävän tuotteen CE-hyväksyntä [15, s. 43].

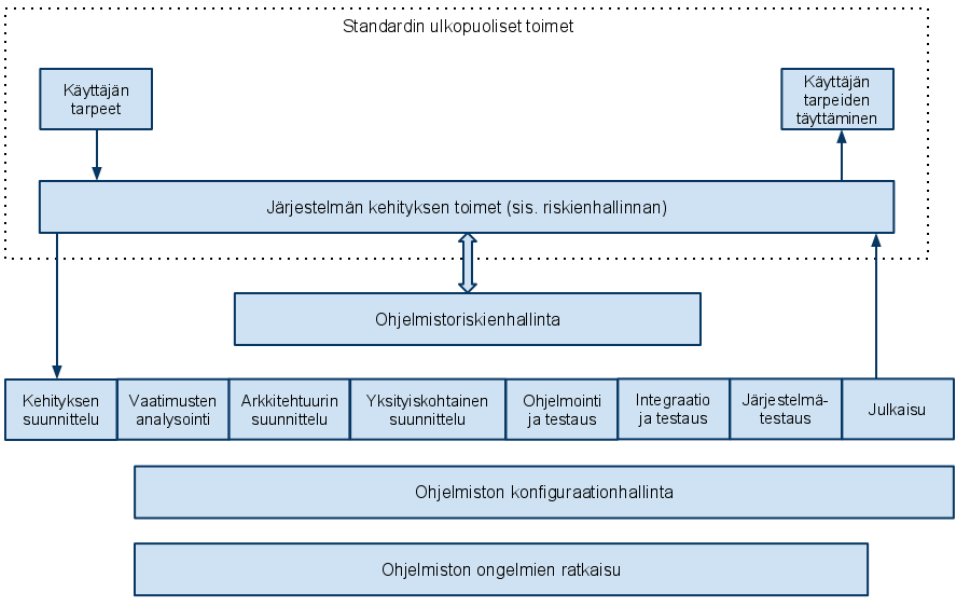
Lääkintälaitteissa olevan ohjelmiston ohjelmistokehitykselle ei ole aikaisemmissa standardeissa esitetty kattavia vaatimuksia EU:n alueella tai Yhdysvalloissa [24, s. 43]. IEC 60601-1 standardi viittaa ohjelmistoihin alakohdassa 1-4, mutta tarkempia ohjeistuksia tai työkaluja ei määritellä vaihejakomallille ja abstraktiotaso on korkea. IEC 60601-1-4 viittaa tarkemmin ottaen ohjelmistoa sisältävää laitteeseen, eikä yksittäiseen ohjelmistoon [15, s. 41], [10, s. 11].

IEC 62304 – medical device software – Software life-cycle process on luotu lääkinnällisen laitteen ohjelmistokehityksen vaihejakomallin kehykseksi. Se on kansainvälinen standardi, joka:

- määrittelee prosessit, jotka kuuluvat ohjelmiston kehityksen elinkaareen,
- sallii eri prosessien käytön vapaasti riippuen ohjelmiston riskeistä,
- sallii jakaa ohjelman eri osiin riippuen toiminnallisuuksien riskiluokista ja

- on harmonisoitu EU:n toimesta.

Standardi kattaa koko ohjelmiston elinkaaren, joten siinä esitellään kehykset sekä kehitys- että ylläpitovaiheelle [24, s. 45-46]. Esitelty elinkaarimalli tai standardi ei kuitenkaan määrittele mitään tiettyä vaihejakomallia, vaan esittää yleiset vaatimukset mallille, jota käytetään ohjelmistokehityksessä. Tämä antaa tuottajalle vapaat kädet valita sopiva vaihejakomalli omaan kehitykseen. Standardin ohjelmistokehityksen toimet on esitetty kuvassa 5.5.



Kuva 5.5: IEC 62304 ohjelmistokehityksen toimet [24, s. 45].

Ohjelmistokehityksen vaihejakomalli on kuvattu erilaisina vaiheina, jotka ovat pitkälti samoja kuin kappaleessa 2 on esitetty. Suunnitteluvaihe on tarkennettu erikseen arkkitehtuuria koskevalla vaiheella, koska ohjelmiston arvioinnin toteuttaminen yksiselitteisesti vaatii arkkitehtuurin suunnittelun ja turvallisuuskriittisten osien huomioimisen. Jokaisella eri vaiheelle on kuvattu vielä vaiheen esivaatimukset ja tuotokset, jotka tulee laatustandardin mukaisesti dokumentoida ja olla jäljitettävissä. Vaikka malli on esitetty standardissa ”putkena”, voi käytettävä vaihejakomalli olla iteratiivinen ja joskus iteroivan mallin käyttö on jopa suositeltavaa [24, s. 45].

Vaihejakomallin lisäksi standardi ottaa kantaa ohjelmiston konfiguraation hallintaan ja ongelmanratkaisuun. Nämä prosessit tukevat ohjelmistokehitystä määrittelemällä versionhallinnan ja ongelmanratkaisujen vaiheet ja toimet, ylläpitämällä ohjelmiston versioita, ohjaamalla muutosten tuomista kehityksen aikana ja doku-

mentoimalla tehdyt toimet. Ohjelmiston ongelman ratkaisu -prosessi käsittää ohjelmointivirheiden, vaatimuksiin ja määrittelyihin liittyvien ongelmien ratkaisun.

Esitelty malli sisältää myös riskienhallinnan ohjelmiston osalta. Malli antaa tuottajalle lähtökohdan riskienhallintaan, joka antaa päätöksenteon perustua oikeaan riskiin eikä tietyn standardin ohjeistuksiin [23]. Standardi jakautuu riskinäkökulmasta kahteen päätoimintoon: 1. Ohjelmiston turvallisuusluokkien (*safety classes*) määritykseen ja 2. vaihejakomallin soveltaminen sisältäen kaikki vaaditut prosessit ohjelmiston turvallisuusluokan mukaan [24, s. 43]. Määritellyt turvallisuusluokat ovat:

- Luokka A: Vamma ei mahdollinen
- Luokka B: Ei vakavaa vammaa
- Luokka C: Kuolema tai vakava vamma mahdollinen

Ohjelmistolle määritellään turvallisuusluokka edellä esitellyistä turvallisuusluokista. Pienin mahdollinen osakokonaisuus, johon ohjelmistoon liittyvä tunnistettu riski voidaan määrittää, on ohjelmiston arkkitehtuurissa esitelty osa (*item*) [23]. Osio voidaan käsittää siis ohjelmistokomponenttina tai jonain loogisena kokonaisuutena ohjelmiston arkkitehtuurissa – kooditasolle ei tarvitse mennä. Tällöin erityisen tärkeään rooliin ohjelmiston kehityksessä ja viranomaisvaatimusten noudattamisessa nousee ohjelmiston arkkitehtuuridokumentaatio. Arkkitehtuuri voi muuttua ohjelmistokehityksen aikana, mikä korostaa riskienhallinta- ja ohjelmistokehitysprosessin yhteisiä töitä. Arkkitehtuurin muuttuessa täytyy ottaa huomioon muutoksesta aiheutuvat tai muuttuneet riskit.

Ohjelmiston turvallisuus voidaan laskea pienempään luokkaan osoittamalla arkkitehtuurista kriittiset komponentit ja muokkaamalla arkkitehtuuria riskien pienentämiseksi [24, s. 45]. Robert Ginsberg esittää, että turvallisen arkkitehtuurin määrittely on yksi tärkeimmistä tehtävistä ohjelmiston riskienhallinnan prosesseista [23, k. 27]. Turvallisen arkkitehtuurin ominaispiirteiksi hän esittää kunnollisen jaottelun, testattavuuden ja ennustettavan toiminnallisuuden.

Riskienhallinnasta IEC 62304 viittaa suoraan ISO 14971:een, joka on esitetty omana prosessinaan tämän standardin ulkopuolella. Riskienhallinnan ja ohjelmistokehityksen vaihejakomallin välille kuitenkin määritellään rajapinta, jonka avulla kontrolloidaan ohjelmistoon liittyviä riskejä. Riskikontrollin toimenpiteet oletetaan konkretisoituvan uusina ohjelmistovaatimuksina [21, s. 9]. Tällöin riskit arvioidaan uudestaan, koska niihin on reagoitu uusina vaatimuksina, jotka pienentävät riskin todennäköisyyttä tai estävät riskiä tapahtumasta [23].

Standardia on kritisoitu siitä, että se ei määrittele rajapintaa tai tehtäviä systeemitason suunnittelulle ollenkaan [24, s. 46-47]. Tämä voi aiheuttaa ongelmia tapauksissa, joissa käsitellään riskejä tai turvallisuuskriittisiä toiminnallisuuksia, jotka koskevat koko tuotetta, eivätkä vain ohjelmiston osaa.

Edellä esitetyn perusteella lääkinnällisen laitteen ohjelmistokehitysmallin tulee olla:

- Dokumentoitu,
- jaettuna eri vaiheisiin,
- vaiheiden esivaatimukset ja tuotokset on määritelty,
- suunnitteluvaiheessa otetaan kantaa eritasoiseen suunnitteluun,
- tuotosten dokumentointi noudattaa käytettyä laatujärjestelmää ja
- riskienhallinta on osa prosessia.

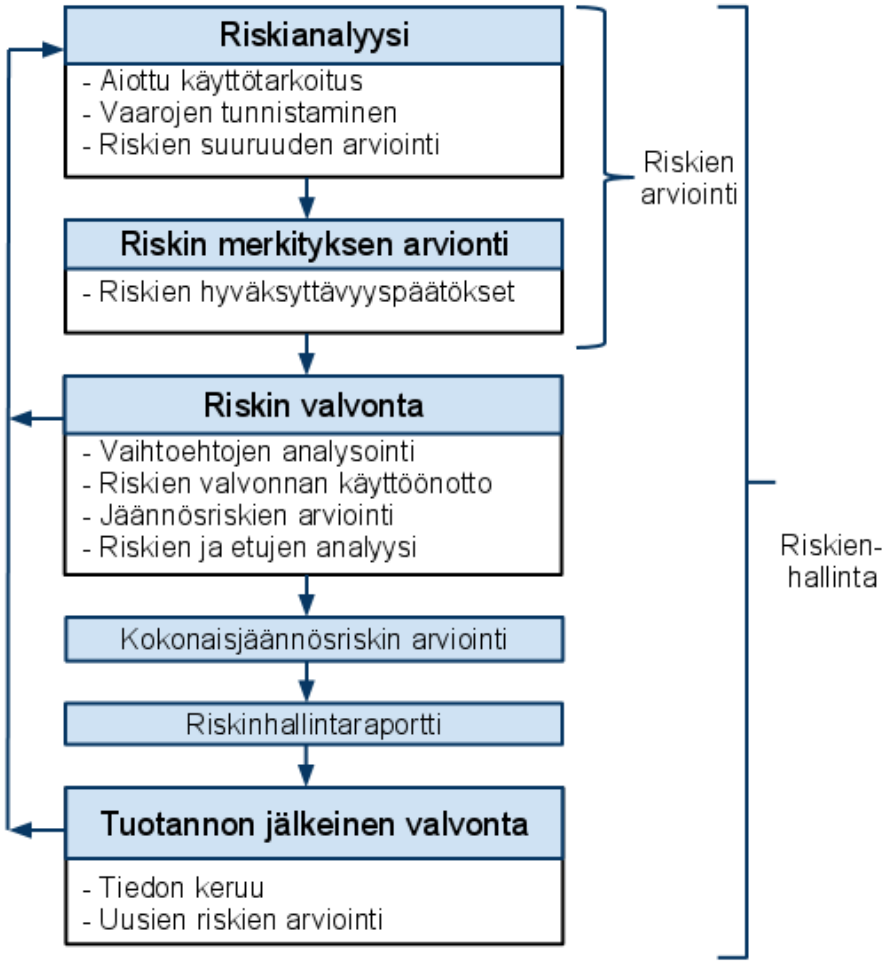
5.7 Riskienhallinta

Lääkinnällisen laitteen riskienhallinta eroaa perinteisen ohjelmistokehityksen riskienhallinnasta siten, että riskit ohjaavat *turvallisen* ohjelmiston suunnittelua. Riskien vähimmäistämistoimenpiteet voivat olla lääkinnällisen laitteen ohjelmistokehityksessä uusia ohjelmistovaatimuksia. Olennainen ero on myös se, että direktiivin tarkoittamat riskit koskevat ensisijaisesti potilaan, käyttäjän tai laitteen läheisyydessä olevan turvallisuutta, eivät projektiriskejä, kuten resursointia. Lääkinnällisen laitteen ohjelmistokehitys voidaan nähdä riskitietoiseksi [11, s. 11].

Riskienhallinta on suunnittelun tukiprosessi, jonka avulla määritellään riskien hyväksyttävät tasot ja menetelmät, joilla asetetut tasot saavutetaan sekä valvotaan näiden riskitasojen noudattamista [16, s. 84]. Lääkinnällisten laitteiden direktiivi viittaa riskianalyysin direktiivin vaatimustenmukaisuusvakuutuksessa (liite II kohta 3.2C) ja tyyppitarkastuksen täyttämisessä (liite III kohta 3.2). Arviointi tehdään harmonisoidun standardin ISO 14971 mukaisesti, mikä tarkoittaa, että tuotteen riskienhallintaprosessin yhdenmukaisuus tarkistetaan riskienhallintaprosessin tuottamista dokumenteista, joita yhteisesti kutsutaan nimellä riskinhallintatiedosto tai riskienhallintakansio [16, s. 121].

Riskienhallintaprosessi voi olla osa suunnitteluprosessia tai se on oma erillinen prosessi [15, s 35]. Vaikka riskienhallinnalla on merkittävä osuus suunnittelun tukena, kattaa riskienhallinta koko tuotteen elinkaaren [40, s. 170], [15, s. 36], [16, s. 84].

Riskienhallinta koostuu neljästä vaiheesta, jotka ovat: riskianalyysi, riskin arviointi, riskin valvonta ja tuotannon jälkeinen valvonta [7, s. 11–14], [5, s. 20]. Riskienhallinnan soveltamisesta terveydenhuollon laitteisiin ja tarvikkeisiin luodun standardin ISO 14971 mukainen riskienhallintaprosessi on esitetty kuvassa 5.6.



Kuva 5.6: ISO 14971 kehys [5, s. 20].

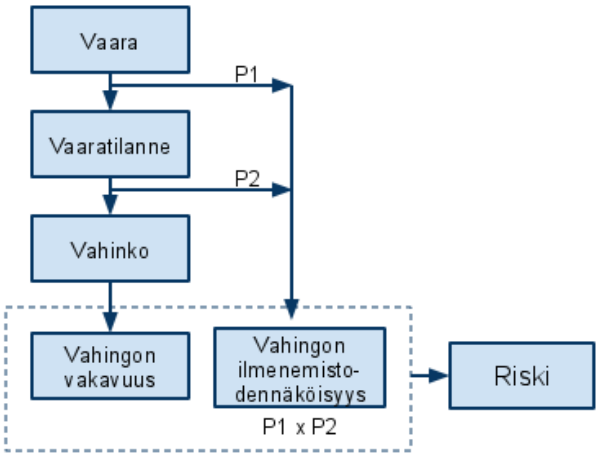
Riskienhallintaa liittyviä termejä standardi ISO 14971 määrittelee seuraavasti:

- **Vahinko**: fyysinen vamma, terveyshaitta tai omaisuusvahinko.
- **Vaara**: vahingon mahdollinen lähde.
- **Vaaratilanne**: olosuhteet, joissa ihmiset, omaisuus tai ympäristö altistuu yhdelle tai useammalle vaaralle.
- **Vakavuus**: mahdollisten vaaran jälkiseuraamuksien mittaaminen.

- **Riski**: vahingon todennäköisyyden ja vakavuuden yhdistelmä.
- **Jäännösriski**: riski, joka jää jäljelle riskinvalvontatoimien suorittamisen jälkeen.

Riskianalyysin aikana tunnistetaan kaikki kohtuullisesti ennalta nähtävät vaarat ja vaaralliset tilanteet mukaan lukien vikatilanteet ja väärä käyttö [16, s. 94]. Kaikkia vaaroja ei ole mahdollista tunnistaa, joten olennaista on tunnistaa tuotteen kannalta tärkeimmät vaaratilanteet [43, s. 492] ja johtaa näihin tilanteisiin kuuluvat vaarat ja vahingot. Analyysin aikana vaaroihin johtavien tapahtumien syyt ja aiheuttajat määritellään sekä tapahtumien todennäköisyydelle annetaan arvio. Vaaratilanteen syy voi olla esimerkiksi laitteisto- tai ohjelmistovika, inhimillinen virhe tai laitteen väärä käyttö.

Vaaran todennäköisyys voidaan ilmaista määrällisesti tai laadullisesti [40, s. 166]. Todennäköisyyden avulla lasketaan riski, joka on siis vaaran vakavuuden ja tämän todennäköisyyden yhdistelmä, esimerkiksi tulo. Ohjelmistovikojen todennäköisyyden määrittely on ongelmallista, koska ohjelmistoviat ovat systemaattisia, eivät sattumanvaraisia, joten niiden todennäköisyys vikaantua tietyllä tapahtumasarjalla on aina joko tosi tai epätosi.



Kuva 5.7: Riskin määrittely [5, s. 100].

Ohjelmistoista aiheutuvien vaarojen todennäköisyydet voi sen sijaan määrittää, kun niiden kuvaamat todennäköisyydet esittävät vaarojen todennäköisyyttä muodostua onnettomuudeksi [16, s. 97-98]. Kuvassa 5.7 on esitetty kuinka riski voidaan määrittää kahden todennäköisyyden tulona. Esimerkiksi todennäköisyyden P1, eli tietyn tapahtumasarjan todennäköisyys johtaa vaaratilanteeksi, oletetaan olevan yk-

si, kun kyseessä on ohjelmistosta aiheutuva vaaratilanne. Vaaratilanteesta aiheutuvan vahingon todennäköisyyttä P2 voidaan pienentää turvallisella suunnittelulla.

Riskianalyysi tarvitsee esitiedoiksi vähintään laitteen aiotun käyttötarkoituksen ja yleisen kuvauksen laitteen toiminnallisuudesta kaikista suunnitelluista käyttötapauksista. Mitä tarkempaa suunnitteludokumentaatiota tai tietoa on käytössä, sitä tarkemmin riskianalyysiä voidaan tehdä. Näistä tiedoista voidaan erilaisilla analyysimetodologeilla tunnistaa ja analysoida riskejä. Tällaisia metodologeja ovat muun muassa:

Alustava riskianalyysi (*preliminary risk analysis, PHA*) on laadullinen tapa tutkia tapahtumaketjuja, jotka voivat muuttaa mahdollisen vaaran vahingoksi. Aluksi tunnistetaan tapahtumat ja ne analysoidaan erikseen. Analyysissa jokaiselle tapahtumalle tunnistetaan mahdolliset aiheuttajat ja ehdotetaan keinoja tapahtuman välttämiseksi. Tällä menetelmällä voidaan tunnistaa suunnittelun ensivaiheissa suurimmat vahingot ja niiden vaaratilanteet. Tätä mallia on kuitenkin kritisoitu siitä, ettei perinteisen alustavan riskianalyysin aikana käytetä tai ole käytettävissä luotettavaa tietoa tuotteesta — kuten suunnitteludokumentaatiota– ja että riskitason arvioinnit ovat hyvin subjektiivisia [49].

Vikapuuanalyysi (*fault tree analysis, FTA*) on ylhäältä alas -tyyppinen (*engl. top-down*) tapa tutkia vaaroja. Menetelmässä otetaan lähtökohdaksi oletetut vaarat ja tämän jälkeen tutkitaan, minkä ehtojen tulee olla voimassa, jotta kyseinen vaara toteutuu. Analyysi muodostaa puumaisen rakenteen ja eri ehtojen yhdistämiseen käytetään loogisia operaattoreita. Vaaralle voidaan laskea todennäköisyys asettamalla puun lehtisolmuille todennäköisyydet ja logiikan laskuoppien mukaan laskemalla todennäköisyys juurisolmulle. Vikapuuanalyysia voidaan käyttää ohjelmistojen analysoinnissa osoittamalla, ettei sen logiikka tuota vikatiloja, jotka ovat aiheuttavat vaaroja [45, s. 570-571], [46, s. 1-2].

Vika- ja vaikutusanalyysi (*failure mode and effects analysis, FMEA*) on formaali tapa kuvata riskejä vika- ja vaikutusnäkökulmasta. Suunnitteluvaiheessa tätä mallia hyödynnetään jakamalla tuote tunnistettuihin komponentteihin tai toimintoihin ja tunnistamalla näille vikaantumistilat. Menetelmä on siis alhaalta-ylös -tyyppinen. Tunnistetuista vikaantumistiloista analysoidaan seuraukset, eli mahdolliset vahingot [48, s. 247]. Lisäksi pyritään kartoittamaan jokaisen vikaantumistilanteen alkuperäinen syy. Myöhemmässä vaiheessa jokaiselle analyysivaiheessa löydetylle vialle arvioidaan todennäköisyys ja vakavuus.

Poikkeamatarkastelu (*hazard and operability study, HAZOP*) on alhaalta ylös -tyyppinen (*engl. bottom-up*) formaali ja systemaattinen tapa tutkia ja tunnistaa suunnitteludokumentaatiosta mahdollisia vaaroja tutkimalla poikkeamia, jotka mahdol-

lisesti johtavat vahinkoon. Tavoitteena on löytää prosessin häiriöistä aiheutuvat vaarat. Poikkeamatarkkailu on alun perin kehitetty kemianteollisuuden tarpeisiin [47, s. 17] ja siitä on tehty sovellutuksia ohjelmistojen riskianalyysiin. Ohjelmistojen analysoinnoissa tutkitaan sisäisen rakenteen tieto- ja ohjausvuota [47, s. 18], [41, s. 47].

Riskianalyysin tuloksena syntyy dokumentoitu esitys tunnistetuista vaaroista, niiden syistä, laajuuksista ja todennäköisyyksistä. Jokaisesta riskistä kuvataan vähintään seuraavat asiat riskianalyysin tuottamalla tavalla kirjattuna:

- **Riskin ID**: jokaiselle tunnistetulle riskille kirjataan yksilöivä tunnistetieto.
- **Vahinko**: yksiselitteinen kuvaus vahingosta.
- **Mahdollinen vaaratilanne**: kuvaus vaaratilanteesta, joka voi aiheuttaa vahingon.
- **Mahdolliset syyt**: jokainen mahdollinen vaaran aiheuttava lähde dokumentoidaan. Lähteitä voi siis olla useita yhdelle vaaralle ja niiden todennäköisyys voi vaihdella.
- **Vakavuus**: vaaran vakavuus kirjataan ylös riskianalyysivaiheen tuloksena.
- **Todennäköisyys**: vaaralle tai tarkemmin sen aiheuttavalle syylle päätelty todennäköisyys.
- **Riskin taso ilman toimenpiteitä**: riskienhallintasuunnitelmassa esitetyn tavan mukaisesti päätelty riskin taso ennen vähimmäistämistoimenpiteitä.

Riskienhallintasuunnitelmassa kuvataan riskien määrittelyihin liittyvät kategoriat, joilla vaarojen vakavuuksia, todennäköisyyksiä ja riskejä voidaan esittää. Näistä kategorioista on esimerkit taulukoissa 5.2 ja 5.3. Riskin hyväksyttävyystaso päätellään taulukosta, johon on todennäköisyyden ja vakavuuden perusteella päätetty eri hyväksyttävyyden tasot. Taulukossa 5.4 on kuvattu yksinkertainen tasomäärittely riskeille.

Riskien merkityksen arviointi tapahtuu riskianalyysin jälkeen kaikille tunnistetuille riskeille. Merkityksen arviointi tapahtuu riskienhallintasuunnitelman mukaisesti ja siihen vaikuttaa analyysissa arvioitu riskitaso, merkittävyys ja riskiin liittyvän toiminnon hyöty [16, s. 100]. Korkean riskitason vaara voidaan siis hyväksyä, jos siitä saatava hyöty on suuri. Jäännösriski arvioidaan tuotekohtaisesti ja se tapahtuu kun kaikki vähimmäistämistoimenpiteet on tehty. Niille riskeille, joille vähimmäistämistoimenpide suoritetaan, kehottaa direktiivi tutkimaan eri ratkaisuja.

Kategoria	*TN*	*Kuvaus*
1	Toistuva	Esiintyy todennäköisesti vähintään kerran kuukaudessa käyttöaikana.
2	Todennäköinen	Esiintyy todennäköisesti korkeintaan 12 kertaa vuodessa käyttöaikana.
3	Satunnainen	Esiintyy todennäköisesti vähintään kerran koko tuotteen käyttöaikana.
4	Harvinainen	Saattaa esiintyä tuotteen käyttöaikana.
5	Epätodennäköinen	Ei todennäköisesti esiinny laitteen käyttöaikana.

Taulukko 5.2: Esimerkki todennäköisyyskategorioista.

Kategoria	*Vakavuus*	*Kuvaus*
1	Katastrofaalinen	Mahdollisesti useita kuolemantapauksia tai vakavia vammautumia.
2	Kriittinen	Mahdollisesti yksi kuolemantapaus tai vakava vamma.
3	Marginaalinen	Mahdollinen vammautuminen.
4	Mitätön	Pieni ärsyke mahdollinen, ei vammautumista.

Taulukko 5.3: Esimerkki vakavuuskategorioista.

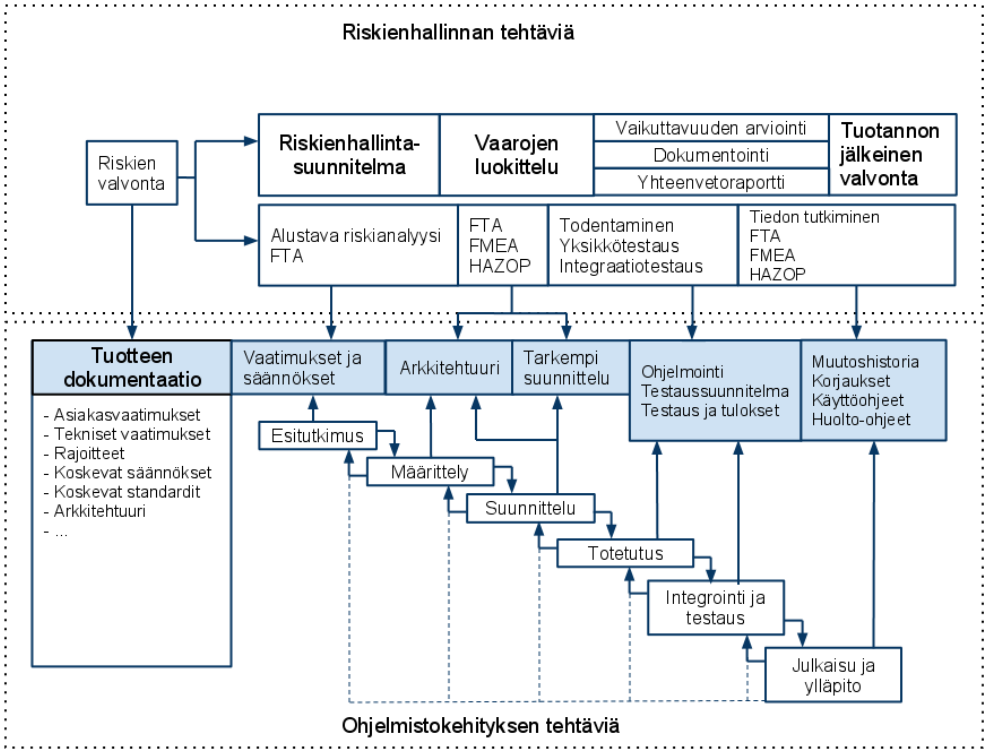
Riskientasot H = Hyväksyttävä, EH = Ei hyväksyttävä, T = Tutki vaihtoehtoja				
TN/Vakavuus	**Mitätön**	**Marginaalinen**	**Kriittinen**	**Katastrofaalinen**
Toistuva	T	EH	EH	EH
Todennäköinen	H	EH	EH	EH
Satunnainen	H	T	EH	EH
Harvinainen	H	H	T	EH
Epätodennäköinen	H	H	H	T

Taulukko 5.4: Esimerkki alustavan riskianalyysin (PHA) riskimatriisista.

Direktiivi ohjeistaa tutkimaan ensiksi voiko riskin aiheuttajan poistaa tai vähentää vaaran todennäköisyyttä turvallisella suunnittelulla. Toissijaisena toimenpiteenä ehdotetaan käyttämään suojauskeinoja, kuten esimerkiksi hälytysjärjestelmiä.

Vähimmäistämistoimenpiteiden jälkeen riskitasot arvioidaan uudestaan ja tulokset dokumentoidaan. Riskianalyysin, suunnitteludokumentaation ja vähimmäistämisen dokumentaatioiden välille tulee selkeä jäljitettävyys siten, että riskianalyysissä on tunnistettu riskit ja vähimmäistämistoimenpiteissä on asetettu vaatimuksia suunnittelulle. Edellä esitetyn ketjun tulee olla jäljitettävissä dokumenteista yksiselitteisesti.

Standardin 14971 mukaan tuotannon jälkeen kerätään systemaattisesti tietoa tuotteen toiminnasta käytössä. Tämän prosessin tarkoituksena on tutkia pitääkö alkuperäinen riskitaso paikkaansa ja tunnistaa uusia riskejä. Uusien riskien löytyessä niiden merkitys tulee arvioida määritellyn riskienhallintaprosessin mukaisesti. Tuotannon jälkeinen tiedonkeruu ja tarkkailu ovat korkeampana käsitteenä osa yrityksen laadunhallintaa.



Kuva 5.8: Riskienhallinnan ja ohjelmistokehityksen tehtäviä [16, s. 86].

Riskienhallinnan ja ohjelmistokehityksen tehtävät suhteessa toisiinsa on esitetty kuvassa 5.8. Riskianalyysia tehdään rinnan ohjelmistokehityksen aikana suunnitteluvaiheessa monella eri tarkkuustasolla ja riskienhallinnan voidaan näin nähdä tu-

kevan tai ohjaavan suunnittelua kohti turvallisempaa ohjelmistoa. Riskienhallinnan toimenpiteiden todentaminen toteutetaan ohjelmistokehityksessä testausvaiheessa. Testaus ja verifiointi nähdään osana riskienhallintaa siten, että vähimmäistämistoimenpiteiden onnistumisen todentaminen on osa riskienhallintaprosessia. Tuotannon jälkeinen riskienhallinta vaikuttaa osaltaan ohjelmiston kehitykseen ylläpitovaiheessa, mutta tehtävät toimet ovat samoja: analysointi, arviointi, vähimmäistämistoimenpiteet ja todentaminen.

5.8 Verifiointi ja validointi

Verifiointi- ja validointitoimenpiteillä osoitetaan, että vaatimukset ovat oikeita, toteutus on oikea ja täyttää vaatimukset. Erityisesti lakisääteisten vaatimusten täyttymisen kannalta nämä toimenpiteet ovat olennaisia, sillä niiden täyttyminen tarkistetaan suunnitteludokumentaation ja validointi-, verifiointi- sekä testiraporttien pohjalta [15]. Verifiointiin ja validointiin otetaan kantaa aikaisemmin esitellyssä riskienhallinnassa ja ohjelmistokehityksessä. Ne määrittelevät osaltaan näiden toimenpiteiden tarpeen, paikan ja tarkkuuden. Samoin ISO 13485 vaatii verifioinnin ja validoinnin toimien kuvauksen. IEEE:n standardi 1012 ohjelmiston verifioinnista ja validoinnista määrittelee termit seuraavasti [6, s. 9]:

- Validointi: Prosessi, jonka avulla osoitetaan, että ohjelmisto ja siihen liittyvät tuotokset vastaavat vaatimuksia jokaisen ohjelmistokehitysvaiheen jälkeen, ratkaisevat oikean ongelman (esim. mallintavat oikein fysiikan lakeja, noudattavat liiketoiminnan malleja, järjestelmään kohdistuvat oletukset ovat oikeita) sekä vastaavat aiottua käyttötarkoitusta ja käyttäjän odotuksia.
- Verifiointi: Prosessi, jonka avulla osoitetaan että ohjelmisto ja siihen liittyvät tuotokset noudattavat vaatimuksia, vastaavat standardeja, käytäntöjä ja merkintätapoja jokaisen ohjelmistokehitysvaiheen aikana sekä onnistuneesti toteuttavat jokaisen elinkaarimallin toiminnon ja täyttävät kaikki kriteerit täyttääkseen elinkaarimallin toiminnon tehtävät (esim. tehdä tuote oikein)

Verifiointi tarkoittaa yksittäisen vaatimuksen oikeaksi todentamista. Verifioinnin voidaan siis katsoa vastaavan kysymykseen: *Teemmekö tuotetta oikein*? Tuotteen verifioinnille tulee laatia suunnitelma, jossa määritellään missä vaiheessa verifiointi toteutetaan, mitkä osat verifioidaan, millä tavoin verifiointi suoritetaan ja milloin tulokset hyväksytään. Verifioinnin tulokset on dokumentoitava. On myös huomattava, että verifioinnin apuna käytetyt työkalut tulee dokumentoida ja tarpeen mukaan

verifioitava ja validoitava. Tällaisia työkaluja voi olla esimerkiksi ohjelmiston automaattisessa testauksessa käytetyt kirjastot tai suunnitelmadokumentaation pohjalta tehdyt tutkimukset, kuten vikapuuanalyysit. Verifiointia voidaan tehdä monessa eri kehityksen vaiheessa, kuten suunnittelun aikana, integraation jälkeen ja ennen ohjelmiston julkaisua. Verifioinnin kohteita ovat esimerkiksi vaatimukset, arkkitehtuuri, suunnitteludokumentaatio, ohjelmistokoodi ja toteutus.

Validointi vastaavaa kysymykseen: *Teemmekö oikeaa tuotetta*? Validointi tarkoittaa lääkinnällisen laitteen tapauksessa tapaa varmistaa suunnitellun tuotteen soveltuvuus aiottuun käyttötarkoitukseen [15, s. 49]. Käsite on siis hieman laajempi verrattuna perinteiseen ohjelmistoon liittyvään validointiin, koska siinä otetaan huomioon laitteen aiottu käyttötarkoitus ja sen asettamat oletukset ja vaatimukset. Validointia suoritetaan julkaistavaksi aiotulle tuotteelle suunnitellussa käyttöympäristössä. Myös yksittäisiä vaatimuksia validoidaan osoittamalla, että ne ovat oikeita. Tällaisia vaatimuksia ovat esimerkiksi erilaiset lääketieteelliset oletukset ihmisen toiminnoista, kuten sykkeen hälytysrajat ja poikkeamat — nämä vaatimukset voidaan osoittaa oikeaksi viittaamalla tutkimuksiin tai asiantuntijan arviointiin. Samoin validoinnissa tarkastetaan, etteivät vaatimukset ole toistensa kanssa ristiriidassa.

Verifioinnissa ja validoinnissa käytettävät tekniikat voidaan jakaa kahteen: tarkastus ja testaus. Tarkastukset ovat staattinen tapa tutkia esimerkiksi vaatimuksia, suunnitteludokumentaatiota tai ohjelmakoodia. Tuloksena on korjausehdotuksia ja huomioita. Tarkastuksia voidaan tehdä usein eri tavoin, kuten esimerkiksi erilaisten tarkistuslistojen tai lukutekniikoiden avulla tai puhtaaseen asiantuntijuuteen luottaen. Testauksessa ohjelmistoon tai sen osaan suoritetaan ennalta määrättyjä testejä ohjelmallisesti tai käsin. Ohjelmiston testaus on dynaaminen ja testaa valmisteilla olevaa tuotetta. Testauksen suunnittelun apuna voidaan käyttää monia eri tekniikoita riippuen testauksen tasosta – tärkeintä on testata ohjelmiston oletettua toimintaa, vikatilanteet sekä väärät arvot, jotta testit ovat kattavia.

Verifioinnin ja validoinnin tulokset dokumentoidaan, siten että ne ovat jäljitettävissä suunnitteludokumentaatioon ja alkuperäisiin vaatimuksiin. Kaikista verifiointi- ja validointitoimenpiteistä koostetaan raportti, joka liitetään osaksi tuotteen teknistä tiedostoa [21, s. 225], [15, s 92].

Lääkinnällisten laitteiden yhteydessä puhutaan myös kliinisestä validoinnista tai kliinisestä arvioinnista [10, s. 28]. Se tarkoittaa kliinistä tutkimusta, joka on tehty lääkinnälliselle laitteelle. Kliininen arviointi täytyy direktiivin mukaan tehdä laitteelle kun laitteella tehdään kliinistä tutkimusta tai hoitoa. EU:n direktiivi 2007/47/EY vaatii kliinisen arvioinnin jokaiselle laiteluokalle. Kliininen arviointi

tehdään tutkimuksella, johon osallistuu oikeita potilaita tai viittaamalla aikaisempiin kliinisiin tutkimuksiin samankaltaisista laitteista tai samankaltaiseen hoitoon perustavan tieteellisen kirjallisuuden analyysiin. Kliinisen validoinnin suunnitelma ja raportointi liitetään osaksi teknistä tiedostoa haettaessa CE-merkintää [15, s 92].

5.9 Yhteenveto

Direktiivin kanssa harmonisoidut standardit asettavat vaatimuksia lääkinnällisen laitteen ohjelmistokehitykselle liittyen ohjelmistokehitysmalliin, suunnitteluun, riskienhallintaan ja verifiointiin. Taulukossa 5.9 on esitetty ohjelmistokehityksen kannalta olennaisimmat direktiivin asettamat vaatimukset.

Taulukko 5.5: Yhteenveto direktiivin asettamista vaatimuksista.

Yleiset vaatimukset, kappale 5.3	*Selite*
Laiteluokka ja aiottu käyttötarkoitus määritelty.	Direktiivin suora vaatimus. Suunnittelun esitiedoksi.
Standardit ja säännökset tunnistettu.	Direktiivin suora vaatimus. Suunnittelun esitiedoksi.
Olennaisten vaatimusten täyttyminen todistettavissa.	Tunnistettujen standardien ja säännösten täyttämisen todentaminen.
Riskienhallintaprosessi käytössä.	Direktiivin suora vaatimus. Ohjaa turvallista suunnittelua.
Suunnitteludokumentaatio on riittävää.	Suunniteludokumentaatiosta löydyttävä kaikki riskienhallinnassa ja vaatimusmäärittelyssä tunnistetut osat.
Tuotteen verifiointi- ja validointitoimet määritelty.	Direktiivin suora vaatimus. Suunnittelun ja toteutuksen todentaminen.
Ohjelmistokehitysmalli on määritelty.	Direktiivin suora vaatimus. Käytettävä ohjelmistokehitysmalli kuvattava.
Ohjelmistokehityksellä rajapinta riskienhallintaan.	Direktiivissä kuvattu riskienhallinta kattaa myös ohjelmiston.
Ohjelmistokehitysmallin vaatimukset, kappale 5.6	
Malli on dokumentoitu.	Mallin jokaisen vaiheen esitiedot, toiminnot ja ulostulot.
Malli on jaettu vaiheisiin.	Malli on jaettava tunnistettuihin vaiheisiin, kuten IEC 62304:n esitys.
Vaiheiden esivaatimukset ja tulokset määritelty.	Jokaiselle vaiheelle on esitiedot ja tulokset.
Suunnittelun eri tasot tunnistetaan.	Suunnittelulle nähdään kaksi tasoa: arkkitehtuuri ja detaljisuunnittelu.
Tulosten dokumentointi noudattaa laatuhallintajärjestelmän ohjeita.	Ohjelmistokehitysmalli toimii järjestelmätason prosessien kanssa.
Riskienhallinta on osa prosessia.	IEC62304 erottaa järjestelmätason ja ohjelmistotason riskit.
Riskienhallintaprosessin vaatimukset, kappale 5.7	
Rajapinnat ja tehtävät määritelty ohjelmistokehitykseen.	Riskianalyysin kautta löydetyt vähimmäistämistoimenpiteet voivat olla uusia ohjelmistovaatimuksia. Jaetut tehtävät tunnistettava.
Riskienhallinta ohjaa suunnittelua.	Riskianalyysi ja riskien vähimmäistämistoimenpiteet toimivat suunnittelun esitietona.
Todentaminen osana kehitystä.	Verifioinnin eri tehtävät ohjelmistokehityksen aikana.

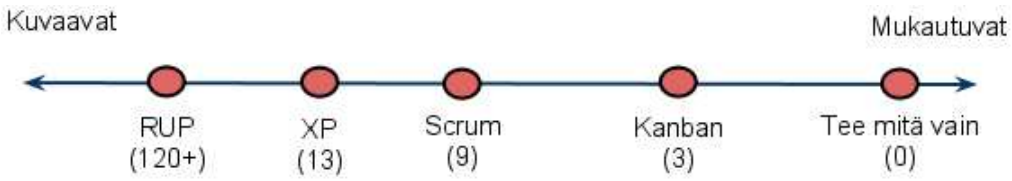
6 Täydennetty Scrum-malli

Tässä kappaleessa esitetään täydennetty Scrum-malli lääkinnällisten laitteiden ohjelmistokehitykseen.

6.1 Mallin valinta ja täydentäminen

Boehm ja Turner esittävät ketteriä menetelmiä arvioivassa kirjassaan [37], että prosessimalleja ei tulisi noudattaa orjallisesti vaan niitä tulee muokata omiin tarpeisiin. Samaa ehdottaa Pöyhönen julkaisussa *"Lääkintälaitteiden ohjelmistot"*, jossa hän esittelee yleisen kehyksen XP-mallin soveltamisesta lääkinnällisen laitteen ohjelmistokehitykseen [10, s. 45]. Samassa julkaisussa hän toteaa, että iteroiden tai prototyyppeinä tehdyt ohjelmiston osat voidaan hyväksyä osaksi valmista markkinoille julkaistavaa tuotetta, mikäli prototyypin kehityksen aikana tehty tuote, dokumentaatio ja käytetty ohjelmistokehitysmalli täyttävät lääkinnälliselle laitteelle ja kehitykselle asetetut vaatimukset [10, s. 56].

IEEE on julkaissut standardin ohjelmistokehityksen vaihejakomalliprosessien kehittämiseen [78]. Siinä kuvataan prosessiarkkitehdin rooli ja tehtäviä prosessimallin kehittämiseksi. Mallin rakentaminen alkaa tunnistamalla tarvittavat vaiheet, näiden esitiedot ja tulokset. Mallia käyttävän organisaation erityispiirteet huomioidaan mallin kehittämisessä, mikäli se on tarpeen. Standardissa ehdotetaan valitsemaan jo määritellyistä ohjelmistokehitysmalleista sopivin tunnistamalla mallien rajoitteet ja vaiheet. Tämän jälkeen mallia täydennetään ja tulos varmistetaan tutkimalla onko siinä kaikki tunnistetut erityispiirteet projektille, jossa uutta mallia käytetään.



Kuva 6.1: Ketterien menetelmien mukautuvuus kuvattujen toimintojen ja roolien mukaan [30, s. 9].

Scrum-mallia pidetään ketteristä menetelmistä erittäin mukautuvana [30, s. 9], joten siihen on helppo määritellä uusia tehtäviä. Mallien mukautuvuus esitettyjen

toimintojen ja roolien määrän mukaisesti on esitetty kuvassa 6.1. XP kuvaa Scrum-mallin tehtävien lisäksi tarkempia käytäntöjä, kuten pariohjelmoinnin ja testivetoisen ohjelmistokehityksen (*test-driven development, TDD*) käytön. RUP (*Rational Unified Process*) puolestaan kuvaa yli 30 roolia, 20 tehtävää ja 70 tulosta eri vaiheille [30, s. 10]. RUP:n haasteena voidaan pitää sen oppimista ja mukauttamista projektiin. Tässä työssä käytettäväksi ohjelmistokehitysmalliksi valitaan Scrum. Scrum on suosiota saaneista malleista mukautuvin, sen käytöstä on hyviä kokemuksia ja se on helppo omaksua [79]. Tutkielman taustalla ei ole organisaatiota, joka vaikuttaisi mallin valintaan. Projektin erityispiirteenä on lääkinnällisen laitteen kehitys.

Scrum-mallia täydennetään kappaleessa 5 esitettyjen lääkinnällisten laitteiden direktiivin ja sen kanssa harmonisoitujen standardien vaatimusten mukaiseksi. Ohjelmistokehitysmallille esitettiin vaatimukseksi dokumentoitu kuvaus mallista, joka on jaettuna selkeisiin vaiheisiin. Vaiheiden esitiedot ja tulokset määritellään sekä suunnittelusta tehdään riittävä dokumentointi. Suunnitteludokumentaatiota voidaan pitää riittävänä kun arkkitehtuuridokumentaatiossa esitellään ohjelmisto jaettuna loogisiin osiin, kriittiset komponentit on tunnistettu sekä suunnitteludokumentaatiosta on viitteet vaatimuksiin ja riskeihin [24], [15, s. 30]. Riskienhallinta on osana ohjelmistokehitystä ja sille on määritelty tehtävät tai rajapinnat ohjelmistokehitysmallissa. Mallissa tulee myös huomioida verifioinnin ja validoinnin tehtävä.

Luvussa 3 esitelty Scrum-malli kuvaa alustusvaiheen, jossa järjestelmän vaatimukset ja arkkitehtuuri kehitetään. Tätä kohtaa on tarpeen tarkentaa suunnittelun ja riskienhallinnan osalta. Alustusvaihetta muokataan luonteeltaan hieman erilaiseksi kuin se on alun perin esitelty. Alustusvaihe eristetään Scrum-mallista ulkoiseksi toimeksi ja Scrum alkaa lyhyellä pyrähdyksellä, jota kutsutaan *0-pyrähdykseksi*. Alustusvaiheen tuloksena ovat asiakasvaatimukset, esitutkimuksen ja vaatimusmäärittelyn tulokset, kuten laitetta ja kehitystä koskevat säännökset, lait ja rajoitteet. Projektisuunnitelma ja alustava riskianalyysi tehdään myös ennen kuin Scrum alkaa. Tällä erittelyllä tavoitellaan parempaa mukautuvuutta ja mahdollistetaan mallin käyttö ulkoistettuun ohjelmistokehitykseen. Kehityksen esitiedot ja vaadittavat alustustoimet tehdään 0-pyrähdyksen aikana.

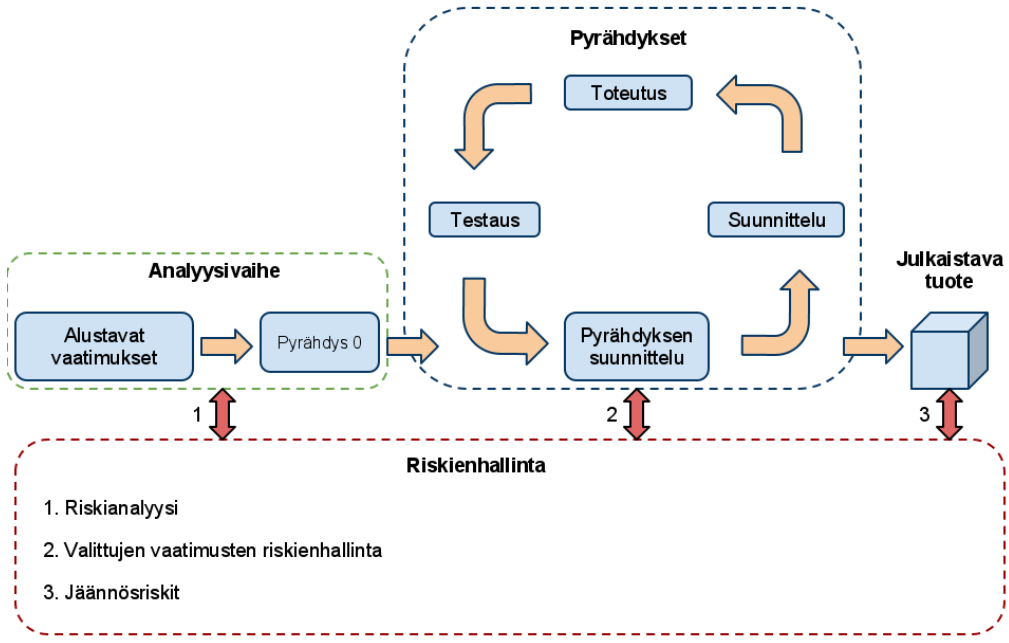
Riskienhallintaprosessille esitetään rajapinnat Scrum-mallin vaiheisiin. Spiraalimallin riskivetoisuutta käytetään pohjana riskienhallinnan ja ohjelmistokehitysmallin integroinnissa. Riskienhallinnan ja ohjelmistokehitysmallin vaiheet ovat seuraavat:

- Riskianalyysi: Projektin alku ja uudet tai muuttuneet vaatimukset pyrähdyksien välissä.

- Riskienhallinta: Pyrähdyksen aikana vähimmäistämistoimenpiteet. Julkaisun jälkeen valvonta.
- Jäännösriskit: Pyrähdyksen lopussa ja julkaisun jälkeen.

Verifiointia ja validointia tarkennetaan kehityksen aikana. Scrum ei määrittele testauksen, verifioinnin tai validoinnin tehtäviä suoraan, joten ne on määriteltävä tarkemmin. Scrum-mallissa on vaatimusten osalta tiettyä validointia, johtuen sen iteratiivisesta ja inkrementaalisesta luonteesta [80]. Pyrähdysten vaihtopalavereissa asiakkaalle näytetään tuotosta uuden toiminnallisuuden osalta, minkä pohjalta hän voi todeta vaatimuksen oikeaksi (validi). Näistä palavereista ei kuitenkaan synny riittävää dokumentaatiota vaatimusten täyttämiseksi. Testausta tarkennetaan lisäämällä jokaisen pyrähdyksen loppuun integraatiotestaus ja lisäksi käyttöön otetaan testivetoinen ohjelmistokehitys XP-mallista sekä esitellään vaatimusten verifiointi.

6.2 Yleiskuvaus

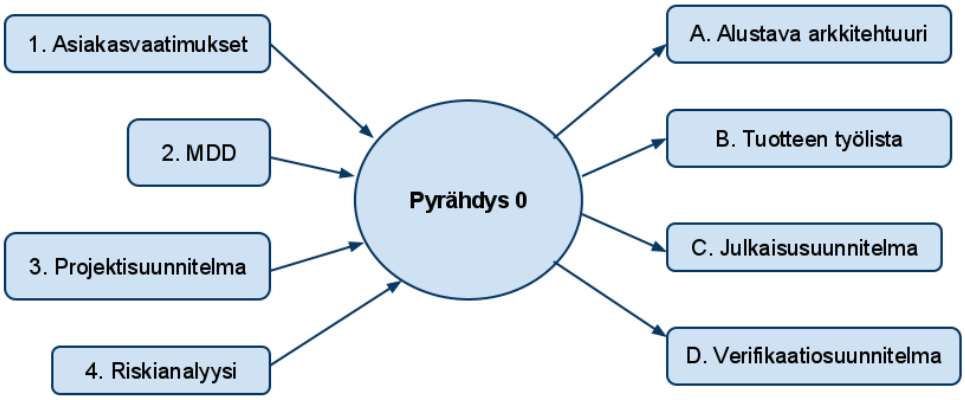


Kuva 6.2: Scrum yleiskuva.

Scrum-malli on jaettu kolmeen pääosaan. Malli alkaa analyysivaiheesta, jossa analysoidaan saadut esitiedot ja vaatimukset. Analyysivaiheen tulokset toimivat toteutusvaiheen pohjana. Toteutus tehdään iteratiivisesti ja inkrementaalisesti pyrähdyksissä kappaleen 3 Scrum-mallin tavoin. Pyrähdykset alkavat aloituspalaverilla,

jossa valitaan toteutettavat vaatimukset ja tutustaan vaatimuksiin liittyviin riskeihin. Pyrähdyksen aikana valvotaan riskejä ja toteutetaan vähimmäistämistoimenpiteet, jotka liittyvät pyrähdyksen vaatimuksiin. Pyrähdys loppuu katselmointipalaveriin ja ohjelmistosta luovutetaan tuotteen omistajalle testattu ohjelmisto ja ohjelmistokehityksen aikana tehty dokumentaatio. Riskienhallinta on Scrum-mallin rinnalla toimiva tukiprosessi, joka jakaa tehtäviä ohjelmistokehityksen kanssa. Kuvassa 6.2 on havainnollistettu Scrum-mallin vaiheet ja rajapinnat riskienhallintaan.

6.3 Analyysivaihe



Kuva 6.3: Pyrähdys 0.

Analyysivaiheen tarkoitus on kartoittaa yleistietoa projektista, estimoida projektille alustava koko ja tuottaa arkkitehtuurikuvaus sekä alustaa riskienhallinnan toimet ohjelmistokehitystä varten. Vaiheen pituutta ei ole määritelty tarkasti, koska tarvittava aika on riippuvainen projektin koosta. Vaiheeseen voidaan olettaa käytettävän vähemmän aikaa kuin yhteen kokonaiseen pyrähdykseen. Pyrähdys nollaa ei ole määritelty aikaisemmin esitellyssä Scrum-mallissa kappaleessa 3 vaan se on tuotu uutena käsitteenä Scrum-malliin, jotta direktiivin asettamat vaatimukset täyttyisivät. Kuvassa 6.3 on esitelty vaihe yleisesti ja taulukoissa 6.1 ja 6.2 on kuvattu pyrähdyksen esitiedot ja tulokset.

Projektin alussa otetaan huomioon direktiivin asettamat vaatimukset ja markkina-alueen erityisvaatimukset [15, s. 12]. Laitteen lääkintälaiteluokka ja aiottu käyttötarkoitus on oltava selvitettynä ennen projektin estimoinnin aloittamista. Lääkintälaiteluokka vaikuttaa projektin aikana tehtävään dokumentaatioon, riskienhallintaan ja testauksen tarpeeseen. Samalla tutkitaan onko kehitettävällä tuotteella vielä jotain erityisvaatimuksia, jotka huomioitava projektin aikana. Tällaisia voivat ol-

Dokumentti	*Selitys*
Asiakasvaatimukset	Vaatimusmäärittelyn pohja. Sisältää vähintään aiotun käyttötarkoituksen, päätoiminnallisuudet, rajoitteet ja laadulliset vaatimukset.
Lääkinnällisten laitteiden direktiivi	Direktiivi tulee ottaa vaatimusmäärittelyn ja suunnittelun lähtötiedoksi. Direktiivi laajentuu aikaisemmin esiteltyihin standardeihin ja niiden noudattamiseen. Samoin tulee ottaa huomioon mahdolliset muut koskevat standardit tai säännökset riippuen markkina-alueesta.
Projektisuunnitelma	Projektisuunnitelmassa määritellään roolit, vastuut, osoitetaan henkilöstön pätevyys ja resurssit.
Alustava riskianalyysi	Riskianalyysi aloitetaan samaan aikaan kuin vaatimusmäärittely. Analyysiä päivitetään 0-pyrähdyksen lopussa analysoimalla tunnistettuja vaaroja arkkitehtuurista.

Taulukko 6.1: Esitiedot pyrähdys 0.

Dokumentti	*Selitys*
Tuotteen työlista	Varsinainen vaatimusmäärittelydokumentaatio esiteltynä Scrum-mallin mukaisesti. Sisältää toiminnalliset vaatimukset ja jokaisella vaatimuksella on prioriteetti, koko, käyttäjätarina-muotoinen esitys, tila, tunniste ja tarvittavat viitteet.
Alustava arkkitehtuuri	Ohjelmiston arkkitehtuurin ensimmäinen versio, joka perustuu tuotteen työlistan käyttäjätarinoihin ja määriteltyihin laadullisiin vaatimuksiin sekä suunnittelun lähtökohdiksi otettuihin riskeihin, säännöksiin ja standardeihin.
Julkaisusuunnitelma	Ohjelmiston julkaisukäytänteet, julkaisuajat ja eri versioiden tarkoitukset.
Verifikaatiosuunnitelma	Suunnitelma siitä, minkä tasoista testausta ohjelmistolle tehdään ja missä vaiheessa määritellyt vaiheet ovat soveltuvia.

Taulukko 6.2: Pyrähdys 0:n tulokset.

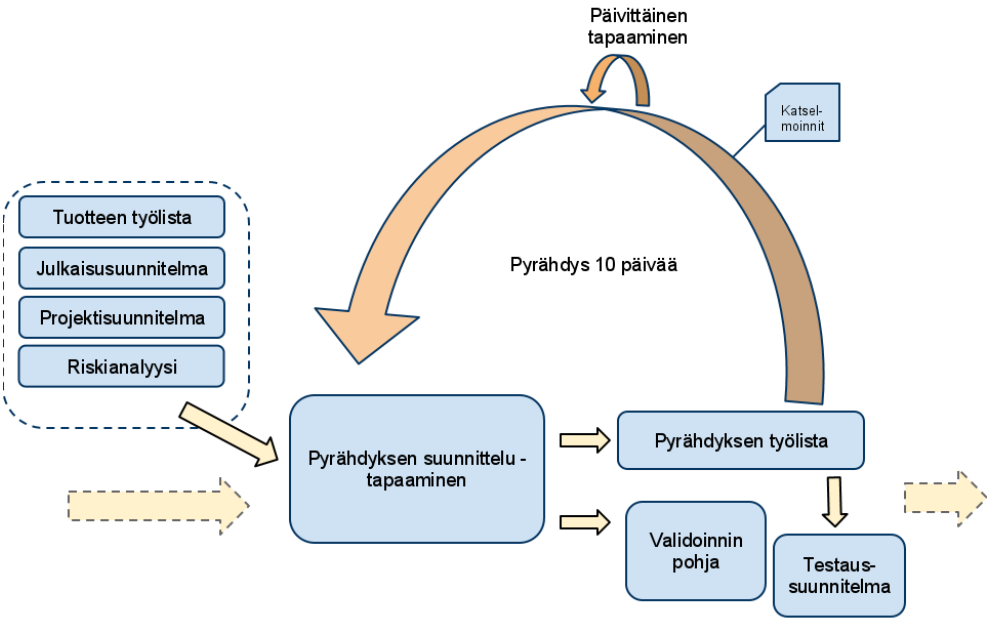
la esimerkiksi käytettävyys- ja hälytysäänistandardit, jotka otetaan tällöin tuotteen suunnittelun lähtökohdiksi [15, s. 29-30], [11, s. 13].

Asiakasvaatimukset ovat kartoitus tuotteen toiminnallisuuksista ja käyttäjien odotuksista. Tästä dokumentista luodaan tarkennettu vaatimusmäärittely tuotteen työlistan muodossa. Lopuksi, kun on saatu riittävän hyvä käsitys tuotteesta, suunnitellaan kehitettävälle tuotteelle alustava arkkitehtuuri. Luvussa 5.6 todettiin arkkitehtuurin olevan riittävä kun siinä on esitetty ohjelmisto loogisina osioina tai komponentteina kriittiset osat tunnistaen. Vähintään nämä elementit kuvataan analyysivaiheessa luotavaan suunnittelun dokumentaatioon. Riskianalyysista voidaan johtaa arkkitehtuurille laadullisia vaatimuksia. Arkkitehtuuri arvioidaan lopuksi laadullisia vaatimuksia ja riskianalyysissa tunnistettuja vaaroja vasten. Arkkitehtuuria korjataan tarpeen mukaan, kunnes se on osapuolien mielestä hyväksyttävä. Syntyvä arkkitehtuurisuunnitelma ei ole viimeinen versio tai ehdoton, vaan sitä päivitetään kehityksen aikana kun ymmärrys halutusta ohjelmistosta kasvaa. Verifiointisuunnitelma luodaan testauksen ja verifioinnin pohjaksi asiakasvaatimuksista ja suunnittelun lähtökohdaksi otetuista säännöksistä.

Projektisuunnitelma antaa puitteet resursointiin ja julkaisusuunnitelman luontiin. Julkaisusuunnitelmaan on valittu tietyt avainominaisuudet eri julkaisupäivämääriin, jotka on määritelty projektisuunnitelman pohjalta. Itse julkaisusuunnitelmasta voidaan viitata suoraan tuotteen työlistaan ja käyttää aikataulutukseen työlistan tehtävien työmääriä. Julkaisusuunnitelmaan kirjataan myös eri versioiden ja versionumeroinnin tarkoitus. Laiteluokkien IIa, IIb tai III ohjelmistoa sisältäviä järjestelmiä ei voida ottaa käyttöön ilman, että ilmoitettu laitos varmistaa järjestelmän kehitys- ja suunnitteluprosessien standardinmukaisuuden kuten luvussa 5 todetaan. Tästä johtuen julkaisusuunnitelmassa tulee ottaa kantaa milloin järjestelmän hyväksyttäminen alkaa ja mitä versioita käytetään tutkimukseen tai muuhun käyttötarkoitukseen, joka ei vaadi CE-merkintää.

6.4 Kehitysvaihe

Kehitys tapahtuu Scrum-mallin mukaisesti pyrähdyksissä. Pyrähdys on esitelty kuvassa 6.4. Yhden pyrähdyksen pituus on noin kaksi viikkoa, jonka aikana tiimi luo mahdollisesti julkaistavan ja käytettävän version tuotteesta. Toteutetut ominaisuudet on valittu pyrähdyksen työlistaan tuotteen työlistasta prioriteettien mukaan. Varsinaiset vaatimukset valitaan ennen pyrähdyksiä pidettävässä suunnittelupalaverissa. Palaverissa tuotteen työlistan prioriteetteja voi päivittää, vaatimuksia lisätä tai poistaa. Priorisoinnin pohjana käytetään tuotteen julkaisusuunnitelmaa ja prio-



Kuva 6.4: Tarkennettu pyrähdysvaihe.

riteeteista vastaa tuotteen omistaja. Pyrähdys alkaa palaverin jälkeen.

Pyrähdyksen alussa luodaan integraatiotestaussuunnitelma ja ennen varsinaista toteutustyötä suunnitellaan jokaiselle pyrähdyksen työlistaan valitulle vaatimukselle testit. Testit toimivat kehityksen ja verifioinnin pohjana. Kehitystä voi kutsua hyväksymistestivetoiseksi, koska jokaista vaatimusta vasten luodaan hyväksymistesti ennen kuin itse toteutusta on olemassa. Tuotoksena syntyy pyrähdyksen testaussuunnitelma ja hyväksymistestit valituille vaatimuksille.

Kehityksen aikana tiimi suunnittelee ja toteuttaa valittuja toiminnallisuuksia itsenäisesti. Tämän aikana voi syntyä mahdollisesti erilaista suunnitteludokumentaatiota. Erityisesti arkkitehtuuridokumentaatiota päivitetään, mikäli siihen tehdään muutoksia. Muuttuneet tai uudet dokumentit julkaistaan pyrähdyksen lopussa.

Riskienhallinta on osa pyrähdystä, joka näkyy riskien vähimmäistämistoimenpiteiden toteutuksena. Joihinkin vaatimuksiin liittyvät riskit vaativat huomiointia suunnittelussa ja tämän tukena toimii riskianalyysi. Tällaisia riskejä hallitaan suunnittelemalla toteutus siten, että kyseiseen vaatimukseen liittyvän riskin todennäköisyyttä pienennetään tai riski käsitellään luvun 5.7 mukaisesti. Joitakin riskejä on mahdoton pienentää turvallisella suunnittelulla. Tällöin riskiä voidaan pienentää kehittämällä esimerkiksi hälytysjärjestelmä, joka ilmoittaa kun riskin toteutumiseen

liittyvä toiminto havaitaan. Riskienhallinnan toimenpiteet käsitellään riskienhallintaprosessin kautta ja tehdyt vähimmäistämistoimenpiteet dokumentoidaan. Vähimmäistämistoimenpiteet konkretisoituvat esimerkiksi uusina vaatimuksina, joten sellaiset vaatimukset on merkittävä vaatimusmäärittelydokumentaatioon. Kaikki syntyvä dokumentaatio on myös katselmoitava ja hyväksyttävä käytettävän laatujärjestelmän mukaisesti. Katselmointien tarve tulee laatujärjestelmä-standardista ISO 13485.

Pyrähdyksen lopuksi syntyy julkaistava tuote tai osa siitä, pyrähdyksen integraatiotestaussuunnitelma, testaustulokset, katselmointien yhteenveto ja muu päivittynyt dokumentaatio. Kehitysvaiheen tulokset on kuvattu tarkemmin taulukossa 6.3.

Tuotos	*Tarkennus*
Ohjelmisto	Käytettävä ja testattu palanen asiakkaan vaatimaa ohjelmistoa.
Testaussuunnitelma	Pyrähdykseen valittujen käyttäjätarinoiden testaussuunnitelma koko ohjelmistolle. Integraatiotestaussuunnitelma.
Testaustulokset	Integraatiotestien tulokset. Tulosraporttiin kirjataan testattu versio, testien kuvaukset ja testaaja.
Katselmointien yhteenveto	Dokumenttien ja koodikatselmointien yhteenveto. Yhteenveto sisältää vähintään katselmoitujen dokumenttien otsikot, versiot ja katselmoinnin tuloksen sekä katselmoijien nimet.
Suunnitteludokumentaatio	Tarkennettu suunnitteludokumentaatio julkaistaan ja ohjelmiston version mukaisesti. Arkkitehtuuridokumentti käsitetään myös ohjelmiston suunnitteludokumentaationa samoin kuin tietokannan kuvaus.
Pyrähdyksen yhteenveto	Yhteenveto tehdyistä käyttäjätarinoista, julkaisupäivämäärä, versionumero ja pyrähdyksen numero.

Taulukko 6.3: Kehitysvaiheen tulokset.

Pyrähdyksen loputtua pidetään pyrähdyksen katselmointipalaveri, jossa esitellään viimeisen pyrähdyksen tulokset. Usein katselmointipalaveri pidetään samaan aikaan seuraavan pyrähdyksen suunnittelutapaamisen kanssa. Tässä palaverissa katselmoidaan toteutus, ehdotetaan mahdolliset muutokset ja hyväksytään tuote. Tämän palaverin aikana voidaan katselmoinnin yhteydessä kevyesti validoida to-

teutetut toiminnallisuudet, mikäli palaveriin osallistuu kehityksestä riippumaton asiantuntija. Validoinnista syntyy tällöin palaverin loputtua oma dokumentti. Validointi koskee tällöin vain tuotteen tiettyjä ominaisuuksia – ei itse tuotetta. Tuote on validoitava kokonaisesta valmiista, julkaistavasta versiosta.

6.5 Verifiointi ja validointi

Ketterissä menetelmissä ja Scrum-mallissa voidaan nähdä suunnittelulle eri tasoja. Hubert Smiths esittelee suunnittelun tasot seuraavasti: tuotteen visio, tuotteen tiekartta, julkaisusuunnitelma, iteraation suunnittelu ja päivittäinen suunnittelu [81, s. 4-9]. Tasot on muutettu kuvaamaan Scrum-mallin termejä taulukossa 6.4 ja niille on esitetty vastuut sekä tavat toteuttaa laatutoimenpiteitä.

Taso	*Vastuu*	*Tapa*
Tuote	Tuotteen omistaja	Laatusuunnitelma
Julkaisu	Tuotteen omistaja	Testaussuunnitelma ja testit
Pyrähdys	Tiimi	Integraatiotestaus, hyväksymistestit ja pyrähdyksen katselmointi
Käyttäjätarina	Tiimi	Hyväksymistestaus ja katselmoinnit
Tehtävä	Tiimi	Yksikkötestaus ja katselmoinnit

Taulukko 6.4: Verifioinnin ja validoinnin tehtävät Scrum-mallissa tasoittain.

Pyrähdys alkaa kirjoittamalla pyrähdykseen valittujen käyttäjätarinoiden pohjalta integraatiotestaussuunnitelma. Testit ajetaan vasta pyrähdyksen lopussa kun kaikki käyttäjätarinat on toteutettu ja integroitu. Testit ja niiden tulokset dokumentoidaan. Dokumentaatiolla osoitetaan verifiointitoimet CE-merkintää haettaessa. Käyttäjätarinoiden työstäminen aloitetaan niin ikään testin kirjoittamisella. Testi voi olla automatisoitu tai manuaalinen. Kehittäminen voidaan nähdä siis hyväksymistestauslähtöisenä kehityksenä. Käyttäjätarinoiden testit ohjaavat kehitystä ja varmentavat toiminnallisuuden kyseisen vaatimuksen osalta. Käyttäjätarinoiden hyväksymistestit eroavat integraatiotesteistä siten, että hyväksymistesti testaa vain yhden vaatimuksen toiminnallisuutta.

Pyrähdyksen aikana tehtävää ohjelmistokehitystä tehdään testivetoisesti. Testivetoinen ohjelmistokehitys ei ole sama asia kuin yksikkötestaaminen, joten kriittisille osille kirjoitetaan myös tarkemmat toiminnallisuutta testaavat testit. Mahdollisten ohjelmistovikojen yhteydessä on suotavaa kirjoittaa automaattinen testi, joka toistaa vian ja vasta sen jälkeen korjataan vika.

Katselmoinnit pyrähdyksen aikana ovat Scrum-tiimin tehtävä. Koodien katselmoinnit ovat yksi tapa verifioida toimintaa ja parantaa ohjelmiston laatua. Vähintään kriittiset osat katselmoidaan ja katselmoinneista dokumentoidaan yleiset tiedot, kuten katselmoijat ja tulokset. Koodin lisäksi katselmoidaan myös hyväksymistestit.

Käyttöön julkaistavalle tuotteelle on joissakin tapauksissa suoritettava testejä pyrähdyksien integraatiotestien lisäksi. Nämä Scrum-mallin ulkoiset testit ovat tuotteen omistajan vastuulla. Testit suoritetaan tuotteen julkaisusuunnitelmassa määritellyille julkaisuille. Tällaisia testejä voi olla esimerkiksi käytettävyystestit tai usean ohjelmiston integraation jälkeiset verifiointitoimet. Lääkinnällisten laitteiden direktiivi vaatii jokaisen julkaistavan ohjelmiston täyttävän sille esitetyt vaatimukset ja CE-merkinnän. Uusia ohjelmistoversioita voi julkaista kevyemmin päivityksinä, mikäli ohjelmisto ei ole muuttunut merkittävästi. Laajempien päivitysten myötä tuotteelle on haettava muutospyyntöä ilmoitetulta laitokselta [15, s. 85-86].

Tuotteen verifiointia ja validointia ohjataan tuotteen omistajan määrittelemällä laatusuunnitelmalla, joka on osa vaadittua laatujärjestelmää [15, s. 14]. Se määrittelee osaltaan testauksen tason aina Scrum-mallin integraatiotesteihin asti, sillä laatusuunnitelman verifioinnin ja validoinnin tasot ovat yhdenmukaisia tuotteen lääkintälaiteluokan kanssa ja kattaa näin koko kehityksen [15, s. 49].

6.6 Riskienhallinta

Riskienhallinnan rajapinta noudattaa luvussa 5.6 esitettyä jakoa. Riskienhallinta pidetään omana prosessina, johtuen siihen liittyvistä tehtävistä, päätöksenteosta ja rooleista. Scrum-mallin roolit pidetään erillään riskienhallinnasta, eikä riskienhallinnan rooleja sekoiteta Scrum-tiimiin. Tiimistä osallistuu riskienhallintaan yksi tai useampi henkilö, jolloin he edustavat riskienhallinnassa ohjelmistoalan erityisosaajia. Erottelu korostaa entisestään ihmisten välistä kommunikointia ohjelmistokehityksen aikana. Riskienhallinnan ja ohjelmistokehitysmallin välille tunnistettiin kolme liittymää, jotka ovat riskianalyysi Scrum-mallin analyysivaiheessa, riskienhallinta pyrähdyksien aikana ja jäännösriskien valvonta pyrähdyksien jälkeen. Ohjelmistoa koskevia riskejä käsitellään Scrumin aikana, joten riskienhallintaprosessilla ja Scrum-mallilla on jaettuja tehtäviä. Tehtävät on ajoitettu Scrum-mallin mukaisesti.

Riskianalyysi alkaa ennen ohjelmistokehitystä tai suunnittelun aikana. Riskianalyysiä voidaan tarkentaa analyysivaiheen 0-pyrähdyksen aikana syntyneen suunnitteludokumentaation tuella. Usein tarkempi riskianalyysi on pakollista suunnit-

teludokumentaation pohjalta, koska riskianalyysitekniikat käyttävät suunnitteludokumentaatiota ohjelmistosta johtuvien riskien syiden tunnistamiseen. Riskianalyysitekniikoita esiteltiin luvussa 5.7 ja niistä ainoastaan alustavaa riskianalyysia voidaan tehdä ilman suunnitteludokumentaatiota. Pelkästään alustavan riskianalyysin käyttö ei ole riittävä, jos laitteen riskitaso on korkea.

Riskienhallinta kattaa koko tuotteen elinkaaren. Riskienhallinnan merkitys on korostunut ohjelmistokehityksen aikana, jolloin uusia toiminnallisuuksia kehitetään. Pyrähdyksien aikana toteutetaan vähimmäistämistoimenpiteitä ja näiden tehokkuus varmistetaan pyrähdyksen lopussa. Scrum-tiimillä on kehityksen aikana oma roolinsa riskien tunnistamisessa ja vähimmäistämisessä, koska tarkempi suunnittelu toteutetaan pyrähdyksen aikana. Riskien arvioinnissa on tunnistettu vaatimuksia koskevat riskit ja niiden vähimmäistämistoimenpiteet. Suunnittelu ja toteutus noudattavat riskienhallinnan tuloksia ja tämän on tultava esiin suunnitteludokumentaatiosta tai verifioinnin tuloksista. Pyrähdyksen lopussa päivitetään riskianalyysin avulla jäännösriskit ja koko tuotteen riskitaso.

Tuotteen työlistalle voidaan lisätä uusia vaatimuksia. Uusiin vaatimuksiin voi liittyä tunnistamattomia riskejä, joten riskianalyysiä tehdään tarvittaessa pyrähdyksien välissä. Tämä hidastaa kehitystä verrattuna perinteiseen Scrum-malliin, johtuen uudesta vaiheesta vaatimuksen kirjauksen ja toteutuksen välissä. Uuden vaatimuksen voi ottaa pyrähdyksen työlistalle vasta kun riskienhallinnan avulla on löydetty vaatimukseen liittyvät riskit ja niiden vähimmäistämistoimenpiteet tai todettu ettei vaatimukseen liity riskejä.

6.7 Yhteenveto

Scrum-mallia täydennettiin luvussa 5 tunnistettujen direktiivin asettamien vaatimusten pohjalta. Malliin lisättiin uusi vaihe, *pyrähdys 0*, jonka aikana suunnitellaan ohjelmiston arkkitehtuuri sekä päivitetään riskianalyysiä ohjelmistovaatimusten ja alustavan arkkitehtuurin pohjalta. Pyrähdys 0:n aikana luodaan ohjelmistolle vaatimukset tuotteen työlistan muodossa ja tehdään julkaisusuunnitelma. Riskienhallinnan tehtäville esiteltiin rajapinnat ohjelmistokehityksen aikana sekä malliin tarkennettiin testauksen tehtävät ja tasot. Täydennetyn Scrum-mallin vaiheille esitettiin selkeät esitiedot ja tulokset. Taulukossa 6.5 on esitetty yhteenvetona mallin vaiheiden esitiedot, tehtävät ja tulokset.

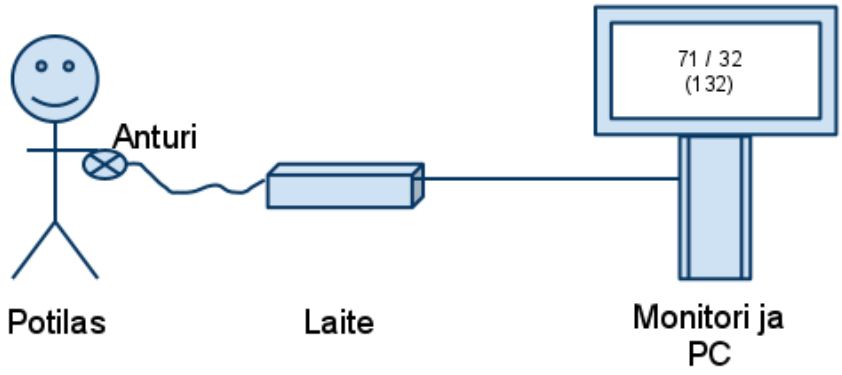
Vaihe	*Esitiedot*	*Tehtävät*	*Tulokset*
Analyysi	Asiakasvaatimukset, projektisuunnitelma, riskianalyysi, tunnistetut standardit ja asetukset	Pyrähdys 0, arkkitehtuurisuunnittelu, riskianalyysi	Arkkitehtuuri, riskianalyysi, tuotteen työlista, julkaisu- ja verifikaatiosuunnitelma
Pyrähdys	Pyrähdyksen työlista, riskianalyysi ja verifikaatiosuunnitelma	Suunnittelu, toteutus, testaus, katselmoinnit, päivittäiset tapaamiset	Ohjelmisto, testaussuunnitelma, testaustulokset, katselmointien yhteenveto, suunnitteludokumentaatio ja päivitykset riskienhallintatiedostoon
Julkaisu	Julkaisusuunnitelma, ohjelmisto ja dokumentaatio	Julkaisu ja ilmoittaminen	Julkaisutiedote ja jäännösriskien päivitys

Taulukko 6.5: Täydennetyn Scrum-mallin vaiheet.

7 Esimerkki

Tässä kappaleessa esitellään kuvitteelliseen lääkinnälliseen laitteeseen liittyvän ohjelmiston ohjelmistokehitys kappaleessa 6 esitellyn Scrum-mallin mukaan toteutettuna.

Järjestelmä, johon ohjelmistoa kehitetään, on ihmisen sykettä mittaava laitteisto. Järjestelmä koostuu anturista, joka kiinnitetään potilaaseen, sekä laitteesta, johon anturi kiinnitetään. Laite sisältää sulautetun ohjelmiston, joka käsittelee anturilta tulevan signaalin. Laitteella on liitäntä ulospäin, jonka kautta se tarjoaa käsitellystä signaalista erilaisia tulkinnallisia arvoja. Kehitettävällä ohjelmistolla on tarkoitus näyttää laitteelta saatava tieto monitorilla sekä generoida hälytyksiä kun arvot ylittävät tai alittavat tietyt arvot.



Kuva 7.1: Yleiskuva järjestelmästä.

7.1 Ennen ohjelmistokehitystä

Ennen PC-ohjelmiston kehityksen aloitusta määritellään projektisuunnitelma. Projektisuunnitelmasta käy ilmi projektin roolit, vastuut ja resurssit sekä rajoitteet. Projektilla on yksi ohjelmistokehityksen tekevä tiimi, tuotteen omistaja ja asiantuntijana toimiva lääkäri. Tiimi koostuu ohjelmoijista, arkkitehdistä ja muista ohjelmistoalan erikoisosaajista. Tuotteen omistajana toimii järjestelmän tilaaja ja hän vastaa viime kädessä vaatimuksista, niiden prioriteeteista ja resursseista.

Esimerkin järjestelmän laiteluokaksi on valittu IIb. Tätä perustellaan sillä, että kyseessä on elintoimintoa tarkkaileva laite. Laite ei aiheuta itsessään minkäänlais-

ta riskiä potilaalle suoran kontaktin kautta. Laite generoi hälytyksiä kun potilaalla tunnistetaan olevan liian korkea tai alhainen syke. Tämän perusteella pahin mahdollinen tapaus potilaalle on kuolema, mikäli laitetta käytetään riskialttiin potilaan[1] ainoana turvana. Aiottu markkina-alue on Eurooppa ja noudatettavaksi säännöksiksi on tunnistettu tutkielmassa luvussa 5 esitetyt standardit sekä hälytysääniä ja käytettävyyttä koskevat standardit EN-457 ja IEC 60601-1-8.

Laitteelle ja koko järjestelmälle on tunnistettu joukko vaatimuksia ja näiden pohjalta on tehty alustava riskianalyysi. Riskienhallinta on aloitettu aikaisemmin sulautetun laitteen kehityksen ja koko järjestelmän esitutkimuksen aikana. Samaa riskitiedostoa käytetään PC-ohjelmiston riskienhallintatiedostona. Järjestelmässä olevan laitteen sulautetun ohjelmiston käyttämä signaalikäsittely-algoritmi on osoitettu tutkimuksella validiksi sekä laitteen rajapinnasta on määrämuotoinen dokumentaatio. Laitteelle on myös tehty tarvittavat mekaniikka- ja sähkötestit CE-merkinnän hyväksyttämiseksi.

7.2 0-pyrähdys

Tuotteen omistaja toimittaa tiimille seuraavat dokumentit: vaatimusmäärittely, projektisuunnitelma, alustava riskianalyysi, laitteen rajapintadokumentaatio sekä tarvittavat säännökset. Tuotteen omistaja perehdyttää tiimin tuotettavaan järjestelmään, sen käyttötarkoitukseen ja esittelee tunnistetut kehitystä koskevat säännökset. Tiimi tutustuu järjestelmän vaatimuksiin ja tekee toimitetusta vaatimusmäärittelystä tuotteen työlistan yhteistyössä tuotteen omistajan kanssa. Tiimi arvioi koot sekä lisää tarvittavat viitteet muihin dokumentteihin. Koot arvioidaan suhteellisen pisteytystavan mukaisesti. Tiimi arvioi jokaisen vaatimuksen työmäärän koon ennalta määriteltyjen pistejoukon mukaisesti. Tässä esimerkissä käytetty pistejoukko on pienimmästä työmäärästä suurimpaan: 0,1,2,3,5,8,13 ja 20. Tässä vaiheessa työlista myös priorisoidaan tuotteen omistajan toimesta. Priorisoinnissa käytetään vapaata numeroarvoa, jonka mukaan vaatimukset laitetaan järjestykseen. Taulukossa 7.1 on esitetty esimerkki tuotteen työlistasta. Taulukon muoto on seuraava:

- **Prioriteetti**: Vaatimuksen prioriteetti numeerisena arvona.
- **Tila**: Vaatimuksen tila, esim.: Ei valmis, Aloitettu, Toteutettu, Testattu, Julkaistu.
- **Vaatimus**: Käyttäjätarina-muotoinen esitys vaatimuksesta.

[1] Esimerkiksi leikkauksesta toipuva potilas.

- **Koko**: Vaatimukseen käytettävän työmäärän arvio. Työmäärää kuvataan suhteellisen pisteytystavan mukaisesti.
- **Viitteet**: Viitteet riskianalyysiin, standardeihin tai suunnitteludokumentteihin. Esim. RI = Riskienhallinta dokumentti.
- **ID**: Yksikäsitteinen tunniste kyseiselle vaatimukselle.

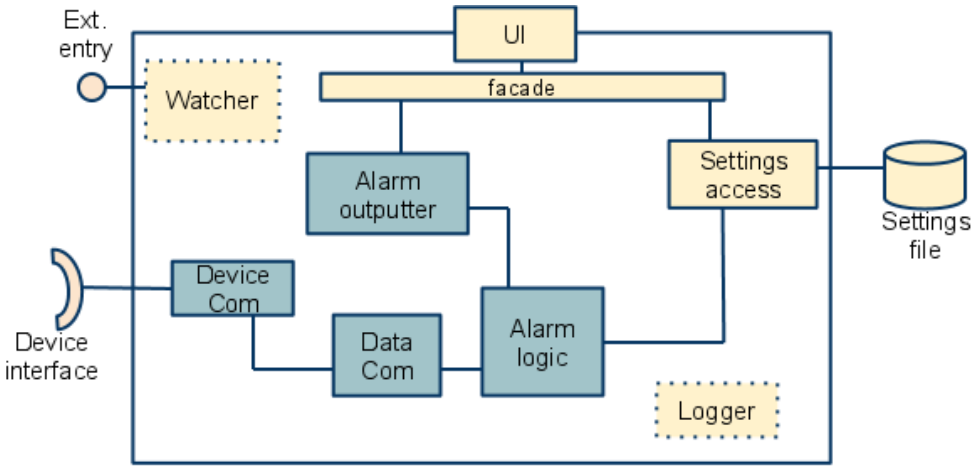
Prioriteetti	*Tila*	*Vaatimus*	*Koko*	*Viitteet*	*ID*
1100	Ei valmis	Hoitajana haluan, että **laite hälyttää äänellä kun syke nousee yli asetetun arvon**, jotta korkea syke huomataan katsomatta monitoriin.	5	RI-2.1 RI-2.3 RI-2.4 EN-475	TT1
1050	Ei valmis	Hoitajana haluan, että **laite hälyttää äänellä kun anturi on irti**, jotta huomaan vian katsomatta monitoriin.	5	RI-3.1	TT2
...	...	...	...	...	...

Taulukko 7.1: Esimerkki tuotteen työlistasta.

Kun tiimi on tutustunut riittävästi järjestelmään ja sen vaatimuksiin, suunnittelevat tiimin jäsenet järjestelmän arkkitehtuurista ensimmäisen version. Kaikki ulkoiset rajapinnat tunnistetaan sekä määritellyt laatuarvot otetaan huomioon. Arkkitehtuurista tunnistetaan ja merkitään kriittiset komponentit riskianalyysin pohjalta. Riskianalyysiä tehdään tunnistettujen vaarojen pohjalta siten, että lähtökohdaksi otetaan yksi tunnistettu vaara ja käydään jokainen arkkitehtuurin komponentti läpi. Mikäli käsiteltävä vaara voi toteutua kyseisen arkkitehtuurin komponentin kohdalla, käsitellään siihen liittyvät vikatilat. Riskianalyysityökaluna voidaan käyttää kappaleessa 5.7 esitettyä vika- ja vaikutusanalyysiä. Riskien viitteet lisätään tuotteen työlistaan niitä koskeviin vaatimuksiin ja riskianalyysidokumenttiin lisätään arkkitehtuurissa käytetyt komponentit. Arkkitehtuuri arvioidaan laadullisten vaatimusten ja riskien pohjalta. Arviointi tehdään laatupuun avulla sekä matriisilla vertaamalla riskianalyysissä tunnistettuja vaaroja arkkitehtuuriin, mistä tulee käydä ilmi, että jokainen vaara on otettu huomioon. Syntyvä arkkitehtuuri antaa järjestelmän

suunnittelulle pohjan ja ohjaa kehitystä. Arkkitehtuuria päivitetään tarvittaessa pyrähdyksien aikana.

Kuvassa 7.2 on esimerkki ohjelmiston arkkitehtuurin suunnittelusta loogisella tasolla. Kriittiset komponentit on kuvattu tummalla värillä. Kuvaa on täydennetty kuvaavalla tekstillä ja siihen on lisätty tarvittavia viitteitä jäljitettävyyden vuoksi. Esimerkissä riskien viittaukset on laitettu tuotteen työlistaan (taulukko 7.1). Arkkitehtuurissa kuvataan komponentit, niiden tehtävät, vastuut sekä viitteet vaatimusmäärittelyyn (esim. TT1) ja suunnittelua ohjaaviin määrittelydokumentteihin (esim DD1).



Kuva 7.2: Palanen arkkitehtuuria.

Komponentti	*Kuvaus*	*Vastuu*	*Viitteet*
Alarm logic	Käsittelee saadun tiedon laitteelta ja päättelee tiedosta onko aiheellista tehdä hälytyksiä. Saa tiedot asetuksista.	Hälytysten generointi ja ohjaaminen ulostuloon.	TT1 TT2 TT3 TT4
Device Com	On suorassa yhteydessä laitteeseen. Käsittelee saadun tiedon ja muuttaa sen järjestelmän sisäiseen muotoon. Ohjaa tiedon eteenpäin järjestelmän sisälle.	Laitekommunikaatio.	TT1 TT2 TT3 TT4 DD1[1]
...	...	...	...

Taulukko 7.2: Järjestelmän komponentit.

Alustavan arkkitehtuurisuunnitelman jälkeen analysoidaan riskejä. Analyysin esitietona on toimitettu riskianalyysidokumentti, josta poimitaan tunnistetut vaarat. Esimerkin riskianalyysissä käytetään vaaroja: ”Ei sykearvoa ruudulla” ja ”Ei hälytystä”. Vaarat listataan jokaista arkkitehtuurin osiota kohden, joten rivejä tulee vähintään vaaran ja osioiden tulon verran. Mikäli vaara ei aiheudu käsiteltävässä osiossa, merkitään riville vain ”Ei sovellu”. Näin voidaan osoittaa, että kaikki arkkitehtuurin osiot on käsitelty. Taulukossa 7.2 on vika- ja vaikutusanalyysillä tunnistettu kuvassa 7.2 esitetyssä arkkitehtuurissa piileviä ohjelmistokohtaisia riskejä. Vika- ja vaikutusanalyysi on yksi esimerkki arkkitehtuurin arvioinnista riskien näkökulmasta.

Riskin vakavuus (*RS*) on esitetty asteikoilla 1—5, jossa 1 on merkittävä riski ja 5 marginaalinen. Riskianalyysin vaiheen tulokset dokumentoidaan järjestelmän yhteiseen riskienhallintatiedostoon, jossa määritellään tunnistettujen vaarojen lisäksi riskien vakavuus ja todennäköisyys. Näillä tiedoilla päätetään toteutetaanko korjausehdotuksia, eli onko riski hyväksyttävällä tasolla vai ei. Vaadittavat korjausehdotukset päivitetään vaatimusmäärittely- ja suunnitteludokumentaatioon. Esimerkissä voitaisiin lisätä uudet vaatimukset yhteyskatkojen ja asetustiedoston virheellisyyden ilmoittamisesta sekä tarkentaa asennustiedostoa käsittelevän arkkitehtuurin osion vastuu tiedon eheyden tarkastamisesta.

Projektisuunnitelman pohjalta tuotteelle tehdään julkaisusuunnitelma tiimin ja tuotteen omistajan yhteistyöllä. Julkaisuversiot merkitään tuotteen työlistaan julkaisuajankohtina, jolloin pyrähdysten suunnittelupalavereissa voidaan julkaisuihin ottaa kantaa helposti. Vain tuotteen omistajalla on oikeus muuttaa julkaisusuunnitelmaa. Laiteluokan IIb ohjelmistoa ei voi suoraan ottaa tuotannossa käyttöön pyrähdyksien lopussa, koska direktiivi vaatii, että koko järjestelmä on verifioitava. Koko järjestelmän verifiointi kattaa esimerkin järjestelmässä anturin, laitteen ja ohjelmiston tuetuissa käyttöjärjestelmissä. Tämän vuoksi julkaisusuunnitelmassa otetaan huomioon koko järjestelmän julkaisutavoitteet.

Tiimi tekee oman verifikaatiosuunnitelman vaatimusten, riskianalyysin, arkkitehtuurin ja laadullisten vaatimusten pohjalta. Tämä dokumentaatio on yksinkertainen verifikaatiotoimien kuvaus. Verifikaatiosuunnitelmaan lisätään yleiset käytänteet kuten ehdot milloin vaatimus on verifioitava ja millä tasolla. Samaan suunnitelmaan kuvataan myös käyttöliittymä- ja käyttäjätestaukset.

ID	*Vaara*	*Osio*	*Toiminto*	*Vikatila*	*Vaikutus*	*RS*	*Korjausehdotus*
RI-3.1	Ei sykearvoa ruudulla	Device Com	Laitekommunikaatio	Yhteys poikki	Arvoja ei saada laitteelta	1	Yhteyskatkosta ilmoitetaan näytöllä ja äänellä
RI-3.2	Ei sykearvoa ruudulla	Data Com	Muuntaa arvot sisäiseen muotoon	Laitteen antamat arvot tulkitaan väärin	Näytettävä arvo lasketaan väärin	3	Ilmoitetaan vääristä laitteen antamista arvoista
...	...	...	...	...	...	...	...
RI-3.13	Ei hälytystä	Alarm logic	Hälytysten generointi	Asetustieto korruptoitunut	Hälytysrajat tulkitaan väärin	1	Lisätään tarkistussumma asetustietoihin, käytetään oletusarvoja vikatilanteissa ja ilmoitetaan korruptoituneesta tiedosta
Ei sovellu	Ei hälytystä	Logger	...	...	...	...	...
...	...	...	...	...	...	...	...

Taulukko 7.2: Vika- ja vaikutusanalyysi arkkitehtuurin avulla.

7.3 Ensimmäisen pyrähdyksen suunnittelupalaveri

Ensimmäisessä suunnittelupalaverissa käydään läpi 0-pyrähdyksen tulokset. Tuotteen omistaja ja muut osapuolet tutustuvat alustaviin suunnitelmiin. Tulokset hyväksytään tai niihin ehdotetaan muutoksia, jotka toteutetaan ensimmäisen pyrähdyksen aikana. Ennen palaveria tiimi on arvioinut nopeutensa, eli kuinka monta pistettä yhden pyrähdyksen aikana voidaan toteuttaa. Tämän pohjalta tuotteen omistaja valitsee pyrähdyksen työlistalle käyttäjätarinoita, jotka toteutetaan ja pyrähdykselle asetetaan tavoite. Tiimi sitoutuu tavoitteeseen. Esimerkissä tuotteen työlistalle on päätynyt samat kaksi käyttäjätarinaa, jotka esiteltiin aikaisemmin. Tuotteen työlistaan muutetaan valittujen käyttäjätarinoiden tila ”Teon alla”.

Pyrähdyksen työlista on samankaltainen kuin tuotteen työlista – siihen on vain listattu pyrähdykseen valitut käyttäjätarinat. Tämä työlista tallennetaan erillisenä dokumenttina, joka nimetään pyrähdyksen numeron mukaan. Palaverin jälkeen tiimi julkaisee pyrähdystiedotteen kaikkien osapuolien saataville. Tiedotteeseen on listattu seuraavat asiat: pyrähdyksen numero, valitut käyttäjätarinat, pyrähdyksen aloitus- ja lopetuspäivämäärä.

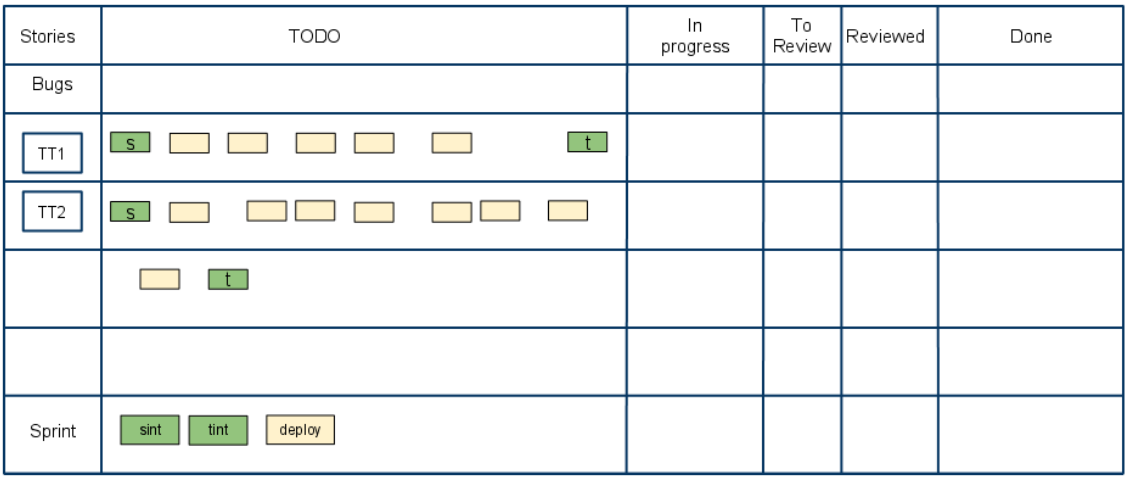
7.4 Pyrähdys

Pyrähdys alkaa pilkkomalla valitut käyttäjätarinat tehtäviksi. Tehtävät kirjoitetaan ylös esimerkiksi post-it -lapuille. Esimerkin käyttäjätarinassa TT1 muutama tehtävä voisi olla: ”Suunnittele hälytysten rajojen tallennus ja luku” ja ”Toteuta hälytysten rajojen tallennus ja luku”. Ensimmäisenä esitetyn tehtävän työn tulos voi olla suoraan arkkitehtuuridokumentaation päivitys. Täydennetyn Scrum-mallin mukaisesti tehtävä menee toteutuksen jälkeen katselmointiin, jolloin vähintään toinen suunnittelija verifioi tuotoksen. Katselmoinneista kerätään lyhyt yhteenveto pyrähdyksen aikana. Tähän tarkoitukseen löytyy versionhallintajärjestelmiin liitettäviä työkaluja[2], jotka automatisoivat katselmointia.

Jokaiselle käyttäjätarinalle tehdään hyväksymistesti ja testin toteutuksen tehtävät. Koko pyrähdykselle tehdään integraatiotestaussuunnitelman ja integraatiotestauksen tehtävät. Näin ollen suunnittelu tulee olemaan testivetoista, koska testi suunnitellaan ennen kuin vaatimukselle tehdään varsinaista toteutusta. Toteutuksen jälkeen toteutus testataan suunnitellun testin mukaisesti. Vaatimus on valmis kun sen hyväksymistesti menee läpi. Pyrähdys alkaa suunnittelemalla integraatiotestaus ja loppuu integraatiotestien ajoon. Kuvassa 7.3 on esimerkki taulusta, johon

[2]Esimerkiksi Reviewboard Subversioniin.

pyrähdyksen tehtävät laitetaan.



Kuva 7.3: Esimerkki tehtävätaulusta.

Tehtävien pilkkomisen jälkeen pyrähdys aloitetaan suunnittelemalla integraatiotesti. Testin pohjaksi otetaan käyttäjätarina ja sen viitteet. Näin ollen testeissä tulee ottaa huomioon vaatimukseen liittyvät riskit tai muut ohjaavat tekijät, kuten standardit ja laatuvaatimukset. Esimerkin vaatimuksessa TT1 viitataan hälytysäänien standardiin, joten testeissä tulee ottaa huomioon, että olennaiset osat on täytetty vaaditusta standardista. Mikäli riskienhallinnassa on jonkin vaatimuksen riskiä päätetty pienentää teknisellä toteutuksella, täytyy toteutus testata ja suunnitteludokumentaatioon lisätään viite riskianalyysiin. Pyrähdyksen aikana voidaan myös huomata uusia riskejä, jotka analysoidaan ja dokumentoidaan riskienhallintaprosessin kautta. Integraatiotestit ajetaan pyrähdyksen lopussa julkaistavalle versiolle. Pyrähdys on julkaisua vaille valmis kun suunnitellut integraatiotestit menevät läpi. Näin ollen testaukselle on hyvä idea varata pyrähdyksen lopusta hieman ylimääräistä aikaa mahdollisten korjausten vuoksi. Alla olevassa tekstissä on esitetty palanen integraatiotestaussuunnitelmasta ja samaisten testien tulosten raportointi on esitelty taulukossa 7.4.

```
Integraatiotesti: IT0101.1
Käyttäjätarina: TT1
Kuvaus: Hälytys ei soi kun syke alle [asetetun arvon].
Testi: 1. Aseta [hälytysrajan arvo] ja [aikaraja]
       2. Simuloi laitteelta syke alle [hälytysrajan] yli
          asetetun [aikarajan]
       3. Varmista, ettei hälytys soi
```

```
       4. Toista kohdat 1-3 hälytysarvoilla 10, 20, 50, 100
          ja 200 aikarajoilla 10, 20, 40, 60
Hyväksymiskriteeri: Kaikki kohdat täytettävä.

Integraatiotesti: IT0101.2
Käyttäjätarina: TT1
Kuvaus: Hälytys ei soi kun syke ylittää [asetetun arvon] alle
        [aikarajan] ajan.
Testi: 1. Aseta [hälytysrajan arvo] ja [aikaraja]
       2. Simuloi laitteelta sykettä yli [hälytysrajan arvon]
          alle asetetun [aikarajan]
       3. Varmista, ettei hälytys soi
       4. Toista kohdat 1-3 hälytysarvoilla 10, 20, 50,100
          ja 200 aikarajoilla 10, 20, 40, 60
Hyväksymiskriteeri: Kaikki kohdat täytettävä.
```

ID	*Kuvaus*	*Huomiot*	*Testaaja*	*Tulos*
IT0101.1	Hälytys ei soi kun syke alle [asetetun arvon]	-	JP	OK
IT0101.2	Hälytys ei soi kun syke ylittää [asetetun arvon] alle [aikarajan] ajan.	-	JP	OK
...	...	...	...	...

Taulukko 7.3: Palanen testausraportista.

Pyrähdyksen viimeinen tehtävä on julkaista ohjelmisto. Tiimi ja tuotteen omistaja ovat aikaisemmin tehneet julkaisusuunnitelman, jossa määritellään eri julkaisujen merkitykset ja versionumerointi. Ohjelmiston lisäksi julkaistaan pyrähdystä koskeva dokumentaatio, eli vähintään päivitetty arkkitehtuuridokumentaatio, katselmointien yhteenveto ja testaussuunnitelma sekä testaustulokset. Scrum-mestari tallentaa pyrähdyskohtaiset dokumentit ja ohjelmiston julkaisun sovittuun paikkaan. Tämän jälkeen tiimi valmistautuu pyrähdyksen katselmointipalaveriin sekä toteutetun ohjelmistopalasen esittelyyn.

7.5 Pyrähdyksen katselmointipalaveri

Pyrähdyksen katselmointipalaverissa käydään läpi viimeisin pyrähdys. Tiimi esittelee pyrähdyksen aikana toteutetut käyttäjätarinat ja järjestelmän toiminnasta otetaan vastaan palaute. Tarkoitus on näyttää ohjelmisto oikeassa toiminnassa ja huomata virheelliset vaatimukset sekä kerätä uusia ehdotuksia järjestelmän toiminnasta. Esimerkin pyrähdyksessä tiimi esittelisi hälytysten toiminnallisuuden, samoin kuten ne on kuvattu integraatiotesteissä.

Järjestelmän esittelyn jälkeen esitellään tuotetut dokumentaatiot ja hyväksytetään ne tuotteen omistajalla. Samassa yhteydessä päivitetään myös riskienhallinnan kautta jäännösriskit, mikäli kyseisessä pyrähdyksessä toteutettiin vaatimuksia, joilla pienennettiin ohjelmistoon liittyviä riskejä. Esimerkissä tällainen on työlistan vaatimus TT2 anturin irtoamisesta: teknisellä toteutuksella pienennettiin riskiä, jossa irtonainen anturi jäisi huomaamatta. Riskienhallintatiedostoon lisätään viite testitapaukseen, jolla kyseisen vähimmäistämistoimenpiteen tehokkuus on varmennettu.

Tuotteen omistaja hyväksyy pyrähdyksen tulokset ja tuotteen työlistalta asetettaan pyrähdykseen valittujen käyttäjätarinoiden tilaksi ”Valmis”. Tämän jälkeen siirrytään seuraavaan pyrähdykseen, joka alkaa jälleen pyrähdyksen suunnittelupalaverilla. Pyrähdyksiä toistetaan kunnes järjestelmä on julkaisusuunnitelman mukaisesti valmis CE-merkinnän hakemiseen. Julkaisusuunnitelma voi ja saa muuttua pyrähdyksien aikana kun odotukset ja visio järjestelmästä tarkentuvat.

Ohjelmistoa koskeva dokumentaatio ja verifiointi ovat osana esiteltyä täydennettyä Scrum-mallia. Tarvittavat toimenpiteet ja dokumentointi CE-merkintää varten on esitelty itse prosessimallissa, joten ne eivät ole osana erillistä dokumentointiprosessia. Jäljitettävyys pysyy hallinnassa iteratiivisesta mallista huolimatta, kun ohjelmiston ja muuttuneiden dokumenttien versionumeroa kasvatetaan jokaisen pyrähdyksen –julkaisun– myötä ja pyrähdysten tuotokset pidetään tallessa.

8 Mallin arviointi

Tässä kappaleessa arvioidaan täydennettyä Scrum-mallia kahdesta näkulmasta: täyttyvätkö direktiivin asettamat vaatimukset ja onko malli vielä Scrum. Direktiivin vaatimuksia käsiteltiin luvussa 5 ja Scrum-malli esitettiin luvussa 3.

8.1 Arvioinnin periaatteet

IEEE:n standardissa ohjelmistokehityksen vaihejakomalliprosessien kehittämisessä esitetään, että kehitetty malli validoidaan niitä vaatimuksia vasten, jotka malliin tunnistettiin kehitysvaiheessa [78, s. 11]. Validoinnin lisäksi standardin mukaan kehitysmallista varmennetaan, että se on kyvykäs viemään ohjelmistokehitysprojektin läpi. Tällä varmennetaan että mallista löytyy kaikki tarvittavat vaiheet ohjelmiston tuottamiseen. Standardi antaa ohjelmistokehityksen vaihejakomalliksi esimerkin iteratiivisesta mallista, jonka vaiheet ovat projektin alustus, suunnittelu, kehittäminen ja käyttöönotto [78, s. 11]. Kehittäminen esitetään iteratiiviseksi ja jokaisen iteraation jälkeen julkaistaan ajettava ohjelma. Standardi tarkentaa dokumentaation tuottamisen ja ohjelmiston testauksen pakollisiksi tehtäviksi projektin aikana.

Standardin perusteella mallia arvioidaan kahdesta näkökulmasta: täyttyvätkö direktiivin asettamat vaatimukset ohjelmistokehitykselle ja onko malli vielä Scrum? Ensiksi tunnistetaan direktiivin asettamat vaatimukset ja tutkitaan täyttääkö malli ne. Toiseksi vertaillaan täydennettyä mallia Scrum-mallin piirteisiin ja ketterien menetelmien periaatteisiin. Kolmanneksi tututustaan aikaisempaan tutkimukseen ketterien menetelmien käytöstä lääkinnällisen laitteen ja riskitietoisen ohjelmiston ohjelmistokehityksessä sekä pohditaan esitetyn mallin käytön haasteita.

8.2 Arviointi vaatimusten näkökulmasta

Direktiivien vaatimuksia käsittelevän luvun yhteenvedossa (kappale 5.9) kuvatut direktiivin asettamat vaatimukset on esitetty taulukossa 8.1 verrattuna täydennettyyn Scrum-malliin. Taulukossa esitetään direktiivin yleiset vaatimukset, ohjelmistokehitysmallin erityisvaatimukset ja ohjelmistokehitykseen liittyvän riskienhallintaprosessin vaatimukset. Vaatimusten täyttyminen suoraan ohjelmistokehitysmal-

Taulukko 8.1: Direktiivin asettamat vaatimukset ja vaatimusten täyttyminen.

Yleiset vaatimukset	*Malli*	*Selite*
Laiteluokka ja aiottu käyttötarkoitus määritelty.	(x)	Analyysivaiheen vaadittu esitieto.
Standardit ja säännökset tunnistettu.	(x)	Analyysivaiheen vaadittu esitieto.
Olennaisten vaatimusten täyttyminen todistettavissa.	x	Vaiheiden tuottama dokumentaatio on kuvattu.
Riskienhallintaprosessi käytössä.	(x)	Riskienhallinta vaadittu tukiprosessi ja riskienhallinnan tehtävät kuvattu.
Suunnitteludokumentaatio on riittävää.	x	Arkkitehtuurin määritelty analyysivaiheessa, muu suunnitteludokumentaatio kehityksen aikana.
Tuotteen verifiointi- ja validointitoimet määritelty.	(x)	Verifiointi määritelty ohjelmistokehityksen osalta, vaatimusten validointi pyrähdyksen vaihtopalavereissa.
Ohjelmistokehitysmalli on määritelty.	x	Esitetty malli luvussa 6.
Ohjelmistokehityksellä rajapinta riskienhallintaan.	x	Riskienhallinnan tehtävät määritelty ohjelmistokehityksen aikana.
Ohjelmistokehitysmallin vaatimukset		
Malli on dokumentoitu.	x	Esitelty luvussa 6.
Malli on jaettu vaiheisiin.	x	Jaettu analyysi-, kehitys- ja julkaisuvaiheisiin sekä vaiheiden tehtävät kuvattu.
Vaiheiden esivaatimukset ja tulokset määritelty.	x	Jokaiselle vaiheelle määritelty vähimmäistiedot ja tulokset.
Suunnittelun eri tasot tunnistetaan.	x	Analyysivaiheen 0-pyrähdys ja pyrähdyksien suunnittelutehtävät.
Tulosten dokumentointi noudattaa laatuhallintajärjestelmän ohjeita.	(x)	Dokumentoitavat tulokset määritelty ja dokumentointi noudattaa ulkopuolista laatujärjestelmää.
Riskienhallinta on osa prosessia.	x	Riskienhallinta on esitetyn mallin rinnalla toimivana tukiprosessina.
Riskienhallintaprosessin vaatimukset		
Rajapinnat ja tehtävät määritelty ohjelmistokehitykseen.	x	Vaiheet ja tehtävät tunnistettu luvussa 6.
Riskienhallinta ohjaa suunnittelua.	x	Riskianalyysi ja riskien vähimmäistämistoimenpiteet toimivat suunnittelun ja pyrähdysten esitietona.
Todentaminen osana kehitystä.	x	Testauksen eri tehtävät ohjelmistokehityksen aikana.

lissa on merkitty x:llä ja epäsuorasti täyttyvät kohdat (x):llä. Epäsuora vaatimus on esimerkiksi säännösten tunnistaminen, jolle ei mallissa suoraan esitetä tehtävää, mutta tunnistetut säännökset vaaditaan esitiedoksi analyysivaiheeseen.

Taulukossa 8.1 kuvattiin direktiivin asettamat vaatimukset ohjelmistokehitysmallin näkökulmasta, mutta dokumentointiin otettiin vain vähän kantaa. Direktiivin vaatimusten täyttyminen tarkistetaan suunnittelun ja kehityksen aikana tuotetusta dokumentaatiosta [11, s. 20]. IEC on julkaissut testiraporttipohjan standardin IEC 60601-1-4 yhdenmukaisuuden testaamiseen [82]. IEC 60601 standardi esitellään luvussa 5 ja standardin alakohtaa 1-4 sovelletaan ohjelmistoa sisältävien laitteiden direktiivin asettamien vaatimustenmukaisuuden tarkistamiseen. Testiraportissa kuvataan vaadittavat dokumentaatiot ja tarkistetaan jäljitettävyys. Taulukkoon 8.2 on kasattu testiraportissa vaaditut ohjelmistokehitystä koskevat dokumentit. Taulukkoon on lisätty esitetyn Scrum-mallin vaihe ja vaiheen tulos, joka vastaa vaadittua dokumenttia.

Dokumentti	*Mallin vaihe*	*Mallin tulos*	*Huomio*
Ohjelmistokehitysmalli	-	-	Mallin kuvaus
Ohjelmiston vaatimusmäärittely	Analyysivaihe, pyrähdyksen vaihtopalaverit	Tuotteen ja pyrähdyksen työlista	Useita versiota, päivittyy pyrähdyksien mukaan
Ohjelmiston arkkitehtuuri	Analyysivaihe, pyrähdys	Arkkitehtuurin kuvaus	-
Suunnitteludokumentaatio	Pyrähdys	Ohjelmistokohtainen suunnitteludokumentti	Tarkempi suunnittelu tapahtuu pyrähdyksien alussa
Testisuunnitelma	Pyrähdys	Integraatiotestisuunnitelma	Tarvittaessa tarkempi testisuunnitelma käytteenotettavalle ohjelmistolle
Suunnitteluympäristön kuvaus	Analyysivaihe	Projektisuunnitelman päivitys	-
Verifikaatiosuunnitelma	Analyysivaihe	Verifikaatiosuunnitelma	Kattaa vain ohjelmiston verifikaation
Muutosten hallinta	Pyrähdys	Päivittyneet dokumentit	-

Taulukko 8.2: IEC 60601-1-4 testiraportin vaatimat tuotokset [82].

Taulukoista 8.1 ja 8.2 nähdään, että direktiivin asettamat vaatimukset on huomioitu täydennetyssä Scrum-mallissa. Riskienhallinta on esitetty osaksi ohjelmistokehitystä, malli on vaiheistettu ja vaiheiden tulokset tukevat verifioinnille ja riskienhallinnalle asetettuja vaatimuksia. Täydennetty malli olettaa, että riskienhallinta ja laadunhallinta ovat erillisiä tukiprosesseja, joten mallin onnistunut käyttö edellyttää edellä esitettyjen prosessien erillistä määrittelyä. Esitettyyn Scrum-malliin on täydennetty tukiprosessien ja ohjelmistokehitysmallin yhteiset toimet. Voidaan todeta, että täydennetty Scrum-malli täyttää direktiivin asettamat vaatimukset lääkinnällisen laitteen ohjelmistokehityksen näkökulmasta.

8.3 Arviointi Scrum-mallin näkökulmasta

Scrum-mallin kuvaavia ominaisuuksia esitettiin Scrum-kappaleen yhteenvedossa luvussa 3.6. Tunnistetut ominaisuudet on esitetty taulukkossa 8.3, jossa esitellään ominaisuuksien kuvaukset ja täydentyminen arvioitavassa mallissa.

Ominaisuus	*Kuvaus*	*Täydennetyssä mallissa*	*Selite*
Joustava julkaisu	Julkaisun sisältö määräytyy ympäristön mukaan	Ei muutosta / Rajoittunut	Direktiivin asettamat vaatimukset ja riskienhallinnan toimenpiteet rajoittavat julkaisua
Joustava aikataulu	Julkaisuaika saattaa muuttua	Rajoittunut	CE-merkin haku, todentaminen ja riskienhallinta hidastavat julkaisua
Katselmoinnit	Edistymistä seurataan päivittäin	Korostunut	Lisätty koodin, suunnittelun ja testien katselmoinnit
Yhteistyö	Sisäinen ja ulkoinen yhteistyö	Korostunut	Riskienhallinta ja toimialan ymmärtäminen
Pienet tiimit	6 henkeä ja useita tiimejä	Ei muutosta	

Taulukko 8.3: Vertailu Schwaberin kuvaamiin piirteisiin.

Samoin luvussa 3.6 esitetyt Knibergin kuvaamat Scrum-toiminnot ja työkalut ovat kaikki edelleen käytössä täydennetyssä Scrum-mallissa. Mallista ei ole poistettu Scrum-mallille ominaisia tehtäviä, joten Knibergin tarkistuslistan mukaan mallin

voidaan edelleen sanoa olevan Scrum. Taulukossa 8.3 esitetyt piirteet ovat mallin arvioinnin kannalta hieman tulkinnanvaraisia. Julkaisun sisällön ja julkaisuaikataulun voidaan nähdä olevan rajoittuneita johtuen ohjelmistokehitystä rajoittavien tekijöiden vuoksi. Tämä ei toisaalta ole mallin suoraan asettama rajoitus ja samat rajoitteet esiintyvät kaikilla malleilla, joilla tehdään lääkinnällisen laitteen ohjelmistokehitystä. Schwaberin Scrum-mallin ominaispiirteiksi kuvattu toistuva katselmointi ja yhteistyö ovat selkeästi korostuneet täydennetyssä mallissa. Katselmoinnin tehtäviä on lisätty ja tarkennettu. Yhteistyön merkityksen todettiin korostuvan riskienhallinnan myötä, koska riskienhallinnalla on korostunut rooli kehityksen aikana. Riskienhallinta ja ohjelmistokehitys on pidetty kuitenkin erillisinä prosesseina, joten ihmisten välinen kommunikaatio korostuu.

Voidaan siis sanoa, että malli on näiden piirteiden kannalta edelleen Scrum. Mielenkiintoisempaa on verrata mallia ketterien menetelmien periaatteisiin. Noudattaako esitetty malli vielä Scrum-mallin taustalla olevia aatteita ja ideologiaa? Ketterien menetelmien arvot on kirjattu Agile manifestoon, joka esitellään kappaleessa 3. Käydään jokainen Agile manifeston esittämä arvo läpi.

Arvo 1: *Arvostamme yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja*. Scrum-mallia täydennettiin uusilla tehtävillä, kuten 0-pyrähdyksellä, testauksella ja riskienhallinnan tehtävillä. Näiden tehtävien noudattaminen nähdään pakollisena direktiivin asettamien vaatimusten täyttämiseksi. Voidaan siis helposti tehdä virheellinen johtopäätös, että prosessin tarkka noudattaminen olisi tärkeämpää kuin yksilöiden ja vuorovaikutuksen merkitys. Yllä esitellyssä Scrum vertailussa todettiin, että yhteistyön merkitys on mallissa korostunut. Riskienhallinta, testien suunnittelu, asiantuntijoiden käyttö ja vaatimusten kehittäminen ovat ihmisten välistä kommunikaatiota. Prosessit ovat vain opastamassa työtä. Voidaan olettaa, että yksilöiden ja vuorovaikutuksen arvostus on edelleen tärkeämpää kuin prosessien pilkun tarkka seuraaminen. Tätä tukee myös Scrum-mallin jälkitarkastelut, joissa tiimillä on vapaus muokata ohjelmistokehityksen kulkua.

Arvo 2: *Arvostamme toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota*. Dokumentaation merkitys on korostunut lääkinnällisen laitteen ohjelmistokehityksessä. Laitteen direktiivin ja säännösten mukaisuus tarkistetaan kehityksen aikana tuotetusta dokumentaatiosta [11, s. 20]. Scrum-mallia täydennettiin uusien tehtävien lisäksi tuotettavalla dokumentaatiolla. Mallin tärkeimpänä tavoitteena voidaan silti pitää toimivan ohjelmiston tuottamista. Tätä oletusta tukevat mallin lyhyet iteraatiot, testaus sekä asiakkaan ja muiden osapuolien välinen tiivis yhteistyö. Dokumentaatio on osa ohjelmistokehitystä ja tukee ohjelmiston turvallisuuden osoittamista sekä markkinoille saattamista. Dokumentaatio ei ole kaikenkattavaa,

vaan ohjattua ja riittävää. On kuitenkin huomattava, että dokumentaation tulee täyttää sille asetetut vaatimukset aivan kuten ohjelmiston. Kysymys on loppujen lopuksi mallia käyttävän organisaation ja ihmisten henkilökohtaisissa arvoissa.

Arvo 3: *Arvostamme asiakasyhteistyötä enemmän kuin sopimusneuvotteluita*. Esitetty Scrum-malli ei ota kantaa sopimusneuvotteluihin. Asiakasyhteistyön rooli on korostunut hieman riskienhallinnan myötä.

Arvo 4: *Arvostamme muutokseen reagoimista enemmän kuin suunnitelman noudattamista*. Esitetyn Scrum-mallin iteratiivista ja inkrementaalista luonnetta ei ole muutettu. Vaatimuksia saa lisätä, poistaa ja muokata edelleen vapaasti. Ainoa rajoite on riskianalyysin tehtävät uusien ja muuttuneiden vaatimusten kohdalla. Tämä hidastaa hieman uuden vaatimuksen siirtymistä kehitykseen ja lisää osaltaan suunnitelmallisuutta.

Mallin voidaan todeta noudattavan vielä Agile manifestossa kuvattuja arvoja. Dokumentaatiota ja prosessia on tarkennettu, mutta ne eivät sulje pois yhteistyön ja ihmisten välisen kommunikaation merkitystä. Yhteistyön merkitys on korostunut entisestään. Esitetty Scrum-malli kuvaa enemmän tehtäviä (mm. testauksen tehtävät ja katselmoinnit) kuin perinteinen Scrum, jonka johdosta esitettyä mallia ei voi pitää yhtä mukautuvana kuin perinteistä mallia. Esitetty Scrum-mallia on täydennetty tehtävillä lääkinnällisen laitteen ohjelmistokehitystä varten, joten mukautuvuuden muutos on oletettua.

8.4 Aikaisempi tutkimus ja mallin haasteet

Ketterien menetelmien ja lääkinnällisen laitteen kehityksestä on tehty hieman tutkimusta. Tutkimukset kohdistuvat haasteisiin, joita ketterät menetelmät kohtaavat turvallisuuskriittisessä ohjelmistokehityksessä. Haasteita asettavat erityisesti laadun osoittaminen, dokumentaation tuottaminen ja jäljitettävyys.

Lin ja Fan esittävät eräänlaisten hybridimallien käytön lääkinnällisten laitteiden ohjelmistokehitykseen [84]. Lääkinnällisen laitteen ohjelmistokehityksessä voidaan käyttää heidän mukaan ketteriä menetelmiä kun huomioidaan kehitystä rajoittavat tekijät ja dokumentoinnin tarkoitus. Vastaukseksi he ehdottavat käytettäväksi ohjelmistokehityksessä ketteriä menetelmiä ja riskienhallinnassa sekä dokumentoinnissa vesiputousmallin kaltaisia prosesseja. Heidän näkemystä tukee Boehmin ja Turnerin kirjoitus jatkuvasti muuttuvien vaatimusten hallinnasta ja ennustettavuudesta [37]. Ketterien menetelmien ja suunnitelmavetoisten mallien välille voidaan asettaa mallit, jotka varautuvat muutoksiin mutta säilyttävät tietyn kurinalaisuuden.

Rottier ja Rodrigues kirjoittavat V-mallin vaihtamisesta Scrum-malliin lääkin-

nällisen laitteen ohjelmistokehitystä tekevässä yrityksessä [20]. Kyseisen yrityksen ohjelmistokehitystä ohjaavat tässä tutkielmassa esitellyt standardit ISO 14971, IEC 60601-1-4 ja IEC 62304. Rottier ja Rodrigues esittävät yrityksen kahteen ohjelmistokehitys projektiin käyttöön Scrum-mallin, jota he ovat täydentäneet jatkuvalla integraatiolla (*continuous integration*), yksikkö- ja hyväksymistestauksella sekä katselmointikäytänteillä. Tässä työssä esitettiin Scrum-malliin myös samat testaus- ja katselmointikäytänteet. Kyseisessä yrityksessä Scrum-malliin siirtyminen nähtiin aluksi ongelmallisena, koska lyhyen iteraatioiden uskottiin olevan ristiriidassa pitkän aikavälin tavoitteiden kanssa. Vanhemman johdon sitoutumisen ja vaikuttamisen myötä Scrum otettiin lopulta käyttöön ohjelmistokehityksessä onnistuneesti. Yhdeksi haasteeksi he kokevat sen, ettei koko organisaatio siirtynyt Scrum-mallin käyttöön. Scrum-mallin käytössä koettiin haasteena turvallisuuskriittisten toimintojen tunnistaminen ja vaatimusten jäljitettävyyden ylläpito. Omat haasteensa toivat toisen projektin vanha ohjelmisto, jota ei voitu yksikkötestata.

Scrum-mallin haasteiksi Rottier ja Rodrigues kokevat tutkielmassaan käyttäjätarinoiden estimointivirheet, hidastavan käsin tehtävän verifioinnin ja projektin valvonnan. Mallissa ei otettu kantaa suunnittelun tasoon, josta myöhemmin seurasi ongelmia järjestelmän arkkitehtuurin kanssa. Vaikka arkkitehtuuri suunniteltiin projektin alussa, sitä ei ylläpidetty tai katselmoitu riittävän usein. Pieneltä vaikuttavia muutoksia lisättiin liian heppoisin perustein, joka myöhemmin johti työlääseen lähdekoodin refaktorointiin. Projektien alussa ei käytetty testivetoistakehitystä, joka myöhemmin todettiin virheeksi. Scrum-mallin hyödyiksi he näkevät ohjelmiston lähdekoodin laadun nousun lyhyiden iteraatioiden ja testivetoisenkehityksen tuloksena. Scrum-malli ei ollut tutkimuksen aikana tehokkaampi kuin aikaisemmin käytetty ohjelmistokehitysmalli, johtuen useiden ohjelmistokehitystä rajoittavien kehysmallien yhteensovittamisesta. Toisaalta Scrum antoi projektien epävarmoissa tilanteissa tukea joustavuutensa vuoksi.

Rasmussen ja kumppanit kirjoittavat ketterän menetelmän käytöstä laiteluokan III lääkinnällisen laitteen ohjelmistokehityksessä laitteissa AgileTek-nimisessä yrityksessä [22]. Yritys siirtyi omaoppisesti ketterien menetelmien käyttöön periaatteinaan lyhyet iteraatiot, automaattinen testaus, päivittäiset tapaamiset ja jälkitarkastelut. Iteraatioiden aikana tuotettiin tarvittava dokumentaation vaatimusten yhdenmukaisuuden todistamiseksi. Iteraatioita seurasi verifiointi- ja validoitintivaihe, jonka aikana ohjelmisto testattiin, suunnitelmat katselmoitiin ja projektin dokumentaatio päivitettiin. Mallia parannettiin myöhemmin kiinnittämällä iteraatiot kuuden tai kahdeksan viikon mittaisiksi ja asettamalla iteraatioihin viikoittainen tavoite, jolla kuvataan toiminnallisia vaatimuksia skenaariopohjaisen tavoitteen sijaan. He tun-

nistivat uuden mallin lyhentävän projektien pituutta ja tarvittavan projektihenkilöstön kokoa noin 20–30 prosentilla. Kokonaiskulujen säästöksi esitetään jopa 35–50 prosenttia. Lyhyempien iteraatioiden ja jatkuvan integraation myötä ongelmat huomattiin nopeammin kuin aikaisemmin käytetyssä vesiputousmallisessa projektissa. He vertailivat lopullisen testauksen tuloksia aikaisemmin vesiputousmallilla kehitettyä ohjelmistoa ketterällä menetelmällä kehitettyyn ohjelmistoon. Ketterällä menetelmällä kehitetty ohjelmisto sisälsi lopullisessa testauksessa merkittävästi vähemmän virheitä kuin perinteisellä mallilla kehitetty.

Fedelmannin ja kumppaneiden tekemässä tutkimuksessa todetaan, että vaatimustenhallintaan liittyvät tehtävät aiheuttavat yli puolet ongelmista lääkinnällisen laitteen kehityksessä [83]. Tämä tulos on yhtenevä ohjelmistokehityksen haasteita kuvaavan CHAOS-tutkimuksen tuloksien kanssa [61]. Fedelmannin tutkimuksessa ei tehty eroa ketteriä menetelmiä ja perinteisiä malleja käyttävien yritysten kanssa. Toinen merkittävä löytö Fedelmannin tutkimuksessa on, ettei projektinhallinnan työkaluja ole riittävästi integroitu osaksi prosesseja. Ketterät menetelmät osaltaan vähentävät vaatimuksissa piilevien virheiden kustannuksia iteratiivisella luonteellaan, jonka hyödyt tunnistettiin myös Rasmussenin ja kumppaneiden paperissa [22]. Tehtävien automatisointia ja työkalujen tärkeyttä ketterien menetelmien apuna korostetaan sekä Rottierin että Rasmussenin löydöksissä.

Tässä tutkielmassa esitettyä Scrum-mallia voidaan verrata Rottierin ja Rodriguesin Scrum-malliin ja tutkimukseen. He eivät esittäneet mallille suoraan rajapintoja eri prosessien tai käytettävien kehyksien tueksi, joka hidasti mallin käyttöä aluksi. Tutkielmassa Scrum-mallia on täydennetty rajapinnalla riskienhallinnan tehtäville ja laadunhallinnan tehtävät on tunnistettu, jonka voidaan nähdä vähentävän mallin mukauttamisesta johtuvaa tutkimustyötä. Yrityksillä voi olla myös muitakin tukiprosesseja ja käytänteitä, jotka rajoittavat ohjelmistokehitystä. Näiden huomiointi asettaa omat haasteensa mallin käytölle. Rottierin ja Rodrigues toteavat testivetoinen kehityksen olevan tärkeää ohjelmiston laadun kannalta. Esitetyn Scrum-mallin yhtenä kuvaavana piirteenä voidaan pitää jatkuvaa verifiointia ja testivetoista kehitystä, koska tarkempaa suunnittelua edeltä aina testien kirjoittaminen ja julkaisua tuotteen verifiointi. Tuotteen verifiointi pyrähdyksen lopussa on ominaista myös Rasmussenin kuvaamalle ketterälle menetelmälle.

Täydennettyä Scrum-mallia voidaan pitää Linin ja Fanin esittämän hybridimallin henkisenä. Malli on kuvaavampi kuin normaali Scrum. Dokumentaatiolle ja riskienhallinnalle on esitetty tehtäviä kehityksen aikana ja riskienhallinta etenee suunnitelmavetoisesti. Ohjelmistokehitys on sen sijaan edelleen ketterien menetelmien periaatteiden mukainen.

8.5 Arvioinnin yhteenveto

Esitetty täydennetty Scrum-malli täyttää direktiivin sille asettamat vaatimukset. Vaatimukset on huomioitu kuvaamalla riskienhallinnan tehtävät ohjelmistokehityksen vaiheissa, tarkentamalla analyysivaihetta sekä esittelemällä testauksen tehtävät kehityksen aikana ja ennen julkaisua. Mallia voidaan pitää edelleen Scrum-mallina ja ketteränä menetelmänä, koska se korostaa ihmisten toimintaa ja käyttää Scrum-mallin tehtäviä. Tapaamisia järjestetään asiakkaan kanssa jokaisen pyrähdyksen yhteydessä, riskienhallinnan tehtävien aikana Scrum-tiimin ja riskienhallinnan osapuolien kanssa sekä päivittäin Scrum-tapaamisissa.

Mallin haasteena voidaan nähdä sen käyttöönotto. Erityisesti sovittaminen organisaation kehityksen tukiprosessien kanssa voi olla haasteellista. Ihmisten välinen kommunikointi on yhä tärkeässä osassa ja heidät tulisi kouluttaa mallin käyttöön hyvin. Mallin onnistunutta käyttöönottoa tukevat jälkitarkastelut, joiden yhteydessä mallia on vapaus muokata omaan käyttöön sopivaksi.

9 Yhteenveto

Lääkinnällisissä laitteissa olevien ohjelmistojen rooli ja merkitys on kasvanut viime vuosina. Potilasvahinkojen myötä viranomaiset ovat alkaneet kiinnittää ohjelmistoihin ja laitteiden suunnitteluun entistä enemmän huomiota. Tämän seurauksesta lääkinnällisten laitteiden myyntiä valvotaan eri markkina-alueilla määriteltyjen säännöksien avulla. Tuotteille suoritetaan säännösten vaatimustenmukaisuuden arviointi ennen myyntiin saattamista. Tässä tutkielmassa säännöksiä tarkasteltiin ETA-sopimuksen määrittelemällä Euroopan talousalueella, jota koskee Euroopan parlamentin ja neuvoston direktiivi 2007/47/EY. Direktiivi asettaa vaatimuksia turvallisuuden näkökulmasta lääkinnällisen laitteen suunnitteluun, kehitykseen ja ylläpitoon.

Vaatimusmäärittely on ohjelmistokehityksen onnistumisen kannalta tärkein vaihe. Tutkimusten mukaan suurin osa virheistä tapahtuu vaatimusmäärittelyn aikana ja samoin on myös lääkinnällisten laitteiden kehityksessä [61], [83]. Käyttäjävaatimusten lisäksi lääkinnällisiä laitteita koskevat aiottujen markkina-alueiden säännökset. Näiden tunnistaminen ja ymmärtäminen on haasteellista, koska lakitekstit ovat usein vaikeaselkoisia ja vaativat toimialan tuntemista. Säännösten lisäksi laitetta voi koskea joukko erityisiä standardeja ja lääketieteellisiä oletuksia, jotka tulee tunnistaa ja varmistaa oikeiksi. Toimialan ymmärtäminen on korostunut ja alan asiantuntijoiden käyttö suunnittelun ja kehityksen aikana on suotavaa.

Tässä tutkielmassa tarkasteltiin direktiivin asettamia vaatimuksia ohjelmistokehityksen näkökulmasta. Lääkinnällisen laitteen ohjelmistot ovat turvallisuuskriittisiä ja tämän vuoksi direktiivi asettaa vaatimuksia ohjelmistojen turvallisuuden näkökulmasta. Riskienhallinnalla on suuri rooli turvallisuuskriittisen ohjelman kehityksessä. Riskejä pyritään ensisijaisesti poistamaan tai pienentämään turvallisella suunnittelulla, joten riskienhallinnan tehtävät osaltaan ohjaavat suunnittelua. Toinen merkittävä tuotteen kehitystä koskeva tehtävä on verifiointi. Testaamalla ja todentamalla osoitetaan, että ohjelmisto toimii oletetusti — myös virhetilanteissa. Riskienhallinnan tehokkuus todennetaan ohjelmistoon ajettavien testien perusteella. Ohjelmistokehitysmallille asetetaan myös ei-toiminnallisia vaatimuksia. Mallin täytyy olla kuvattuna ja jaettuna eri vaiheisiin. Vaiheiden esitiedot ja tuotokset täytyy olla kuvattuna. Riskienhallinta on osa ohjelmistokehitystä ja tämän täytyy tulla ilmi mallin kuvauksesta.

Tutkimuksen tavoitteena oli osoittaa, että ketterää Scrum-ohjelmistokehitysmallia voidaan käyttää lääkinnällisen laitteen ohjelmistokehityksessä. Scrum-mallia pidetään mukautuvana ketteränä ohjelmistokehitysmallina. Se kuvaa verrattain vähän tehtäviä ja rooleja. Malli on kuitenkin sellaisenaan riittämätön lääkinnällisen laitteen ohjelmistokehitykseen, koska se ei määrittele suunnittelua riittävällä tasolla eikä ota kantaa riskienhallintaan tai verifiointiin. Vaatimusten täyttämiseksi malliin lisättiin riskienhallinnalle rajapinta ja yhteisiä tehtäviä sekä määriteltiin analyysivaihe ennen ohjelmistokehitystä ja tarkennettiin testauksen tehtävät. Scrum-mallia täydennettiin lisäksi vaiheiden esitiedoilla ja tuloksilla.

Täydennettyä mallia arvioitiin kahdesta näkökulmasta: täyttyvätkö direktiivin asettamat vaatimukset ja onko malli vielä Scrum? Mallia on täydennetty huomioiden kaikki tunnistetut ohjelmistokehitystä koskevat vaatimukset, joten se täyttää direktiivin asettamat vaatimukset. Lääkinnälliselle laitteelle tehtävä vaatimustenmukaisuuden tarkistus tehdään tutkimalla kehityksen aikana tuotettua dokumentaatiota. Vaatimusmäärittelyn dokumentaatio ja riskienhallinnan tulokset ovat tarkistuksessa tärkeässä asemassa. Vaatimukset ja tunnistetut riskit täytyy olla jäljitettävissä kaikkialle suunnitteludokumentaatioon ja verifioinnin tuloksiin. Dokumentaatiota on tarkennettu mallin vaiheisiin esitiedoiksi ja tuloksiksi.

Täydennetyn mallin tunnistettiin edelleen olevan Scrum, koska se noudattelee Scrum-mallin toimintoja. Kuvaavia ominaisuuksia Scrum-mallille on pienet tiimit, joustava julkaisu ja aikataulu sekä katselmoinnit ja yhteistyö. Lääkinnällisen laitteen kehitys asettaa rajoitteita julkaisun suhteen, johtuen verifioinnin tehtävistä, mutta julkaisun oletetaan olevan edelleen joustavaa mallin iteratiivisesen ja inkrementaalisen luonteen vuoksi. Korostunut prosessin noudattaminen ja dokumentaation merkitys lääkinnällisen laitteen ohjelmistokehityksessä voidaan nähdä olevan ristiriidassa ketterien menetelmien periaatteiden kanssa. Agile manifestossa korostetaan prosessin seuraamisen ja dokumentaation sijaan ihmisten välistä yhteistyötä ja toimivaa ohjelmistoa. Ihmisten välinen kommunikaatio on edelleen korostunut täydennetyssä mallissa, johtuen uusista riskienhallinnan tehtävistä. Dokumentaatio ei ole mallin päämäärä, vaan turvallisen ja toimivan ohjelmiston luonti. Dokumentaatiota tehdään kehityksen aikana turvallisuuteen liittyvistä tehtävistä, mikä osaltaan tukee kehitystä, muutosten arviointia turvallisuuskriittisiin osiin ja tuotteen direktiivin vaatimustenmukaisuuden arviointia. Ketterien menetelmien arvot ovat loppujen lopuksi ihmisten arvoissa ohjelmistokehityksen aikana. Täydennetyn Scrum-mallin todettiin edelleen olevan ketterä menetelmä, joskin se on kuvaavampi kuin perinteinen Scrum-malli.

Mallin haasteeksi todettiin sen sovittaminen muiden erityisten prosessien tai ke-

hyksien kanssa, joita käytetään lääkinnällisen laitteen suunnittelun ja kehityksen tukena. Malli on edelleen verrattain mukautuva, joten sitä voidaan täydentää edelleen uusilla tehtävillä. Haasteena on oikeiden vaiheiden ja tehtävien tunnistaminen. Muutoksia esitettyyn malliin voidaan tuoda jälkitarkastelujen kautta.

Olisi mielenkiintoista tutkia tässä työssä esitetyn täydennetyn Scrum-mallin käyttöä ohjelmistoprojekteissa. Ketterien menetelmien käytöstä lääkinnällisen laitteen ohjelmistokehityksessä on muutamia tutkimuksia, joissa malleja on täydennetty osin samoin tehtävin kuin tässä tutkielmassa esitettyä mallia [22], [20]. Näiden mallien on todettu sopivan tehtävään hyvin ja joissakin tapauksissa mallin käyttö on vähentänyt kuluja ja vähentänyt ohjelmiston toteuttamiseen käytettyä aikaa verrattuna perinteisiin suunnitelmavetoisiin ohjelmistokehitysmalleihin [22]. Direktiivin ohjelmistokehitysmallin vaatimustenmukaisuuden osoittamisessa käytettävässä standardissa IEC 62304 esitetään, että ohjelmistokehitysmalli voi olla iteratiivinen ja inkrementaalinen. Ohjelmistokehitystä ei nähdä vesiputousmaisena vuona, vaan tiettyjen tehtävien toistuvana syklinä. Tämä osaltaan kannustaa tutkimaan ja soveltamaan ketterien menetelmien käyttöä lääkinnällisten laitteiden ohjelmistokehityksessä.

Lähteet

[1] Euroopan parlamentti ja neuvosto, *Euroopan parlamentin ja neuvoston direktiivi 93/42/ETY*, saatavilla WWW-muodossa <URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31993L0042:Fi:HTML`>, viitattu 30.9.2009

[2] Euroopan parlamentti ja neuvosto, *Euroopan parlamentin ja neuvoston direktiivi 2007/47/EY*, saatavilla WWW-muodossa <URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2007:247:0021:01:FI:HTML`>, viitattu 30.9.2009

[3] Schnoll L., *The CE Mark: Understanding the Medical Device Directive*, Paton Professional, California, 2007

[4] IEC, *Medical electrical equipment: part 1-4: General requirements for collateral standard: programmable electrical medical systems*, IEC Std 60601-1-4, 2000

[5] ISO, *Medical devices – Application of risk management to medical devices*, ISO Std 14971:2007, Suomen standardisoimisliitto, Helsinki, 2007

[6] IEEE, *IEEE Standard for Software IEEE Standard for Software Verification and Validation*, IEEE Std 1012-2004, 2004

[7] ISO, *Risk management Guidelines on principles and implementation of risk management*, ISO Std 31000:2009, 2009

[8] Schwaber K., Beedle M. *Agile Software Development with Scrum*, Prentice Hall, New Jersey, 2001

[9] Schwaber K., *SCRUM Development Process*, OOPSLA'95 Workshop on business object design and implementation, Springer-Verlag, New York, 1995, s. 1–23

[10] Pöyhönen I., *Lääkintälaitteiden ohjelmistot*, VTT Tiedotteita 2320, VTT Publications, Espoo, 2006

[11] Pöyhönen I., Hukki K., *Riskitietoisen ohjelmiston vaatimusmäärittelyprosessin kehittäminen*, VTT Tiedotteita 2263, VTT Publications, Espoo, 2004

[12] Atwood J., *Chickens, Pigs, and Really Inappropriate Terminology*, Blog-kirjoitus, 16.10.2006, saatavilla WWW-muodossa <URL: http://www.codinghorror.com/blog/archives/000704.html>, viitattu 19.7.2010

[13] Schwaber K., *Scrum Guide*, saatavilla WWW-muodossa <URL: http://www.scrumalliance.org/resources/598>, viitattu 30.9.2009

[14] Ruotsin Lääkintälaitos, *Proposal for guidelines regarding classification of software based information systems used in health care*, saatavilla WWW-muodossa <URL: http://www.lakemedelsverket.se/upload/foretag/medicinteknik/en/Medical-Information-Systems-Report_2009--06-18.pdf>, viitattu 26.10.2009

[15] Pöyhönen I., Kylmälä K., *EU:n lääkintälaitedirektiivin mukausten terveydenhuollon tuotteiden suunnitelu ja valmistus viranomaisvaatimukset huomioiden*, VTT Publications, Espoo, 2010

[16] Pöyhönen I., Kylmälä K., Harju H., Kemppainen-Kajola P., Kuhakoski K., Spankie G., Ventä O., *Vaatimukset ohjelmistoa sisältäville lääkintälaitteille - Hallinta ja menetelmät vaatimustenmukaisuuden osoittamiseksi*, VTT Tiedotteita 2150, VTT Publications, Espoo, 2002

[17] Eisner L., Brown L., Modi D., *A Primer for IEC 60601-1*, MDDI, Volume 9, 2003, saatavilla WWW-muodossa <URL: http://www.devicelink.com/mddi/archive/03/09/015.html>, viitattu 28.12.2009

[18] Kylmälä K., *IEC 60601-1 ja riskienhallinta osana tuotekehitysprosessia*, kalvosarja, VTT Publications, Espoo, 2009

[19] Beck K. et al., *Manifesto for agile software development*, saatavilla WWW-muodossa <URL: http://agilemanifesto.org>, viitattu 20.1.2010

[20] Rottier P., Rodrigues V., *Agile Development in a Medical Device Company*, Agile 2008 Conference, IEEE Computer Society, Los Alamitos, 2009, s. 218–223

[21] Faris T., *Safe and sound software – Creating and Efficienf and Effective Quality System for Software Medical Device Organizations*, SQ Quality Press, Milwaukee, 2009

[22] Rasmussen R., Hughes T., Plaines D., Jenks J., Skach J., *Adopting Agile in an FDA Regulated Environment*, Agile 2009 Conference, IEEE Computer Society, Los Alamitos, 2009, s. 151–155

[23] Ginsberg R., *The role of Risk Management in IEC/ISO 62304*, kalvosarja, Webinar, 2010

[24] Jorda P., *Standard IEC 62304 – Medical Device Software – Software Lifecycle Processes*, The Institution of Engineering and Technology Seminar for Software for Medical Devices, 2006, s. 41–47

[25] Boggs R. *The SDLC and six sigma, an essay on which is which and why?*, Issues in Information Systems, Volume 5, Issue 1, 2004

[26] U.S Department of Transportation, *Systems Engineering for Intelligent Transportation Systems*, 2007, saatavilla WWW-muodossa <URL: `http://www.ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf`>, viitattu 24.6.2010

[27] U.S Department of Defence, *Systems engineering fundamentals*, Defense acquistition university press, Fort Belvoir, Virginia, 2001

[28] V-Modell XT authors, *Fundamentals of the V-Modell*, versio 1.3, saatavilla WWW-muodossa <URL: `http://v-modell.iabg.de/dmdocuments/V-Modell-XT-Gesamt-Englisch-V1.3.pdf`>, viitattu 2.6.2010

[29] Boehm B., *A spiral model of software development and enhancement*, IEEE Computer, Volume 21, Issue 5, 1988, s. 61–72

[30] Kniberg H., Skarin M., *Kanban and Scrum – making the most of both*, C4Media, Kristianstad, 2010

[31] Otto P., Antón A., *Addressing Legal Requirements in Requirements Engineering*, 15th IEEE International Requirements Engineering Conference, IEEE Computer society, Los Alamitos, 2007, s. 5–14

[32] Siena A., Mylopoulos J., Perini A., Susi A., *From Laws to Requirements*, 2008 Requirements engineering and law (RELAW'08), Barcelona, 2008, s. 6–10

[33] Breaux T., Antón A., Boucher K., Dorfman M., *Legal Requirements, Compliance and Practice: An Industry Case Study in Accessibility*, 16th IEEE International Requirements Engineering Conference, 2008, s. 43–52

[34] Wiegers K., *Software Requirements, Second Edition*, Microsoft Press, Redmond, 2003

[35] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990, Software Engineering Standards Committee, New York, 1990

[36] Deemer P., Benefield G., *The Scrum primer – An introduction to agile project management with Scrum*, goodagile, saatavilla WWW-muodossa <URL: http://www.goodagile.com/scrumprimer/scrumprimer.pdf>, viitattu 2.7.2010

[37] Boehm B., Richard T., *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley, Massachusetts, 2004

[38] Kohler M., *Commission communication in the framework of the implementation of the Council Directive 93/42/EEC of 14 June 1993 concerning medical devices*, Official Journal of the European Union, 2009, s. 39–68

[39] Kessler L., *Summary Technical Documentation for Demonstrating Conformity to the Essential Principles of Safety and Performance of Medical Devices (STED)*, 21.2.2008, saatavilla WWW-muodossa <URL: http://www.ghtf.org/documents/sg1/sg1final-n11.pdf>, viitattu 6.7.2010

[40] Bartoo G., *Risk Management*, IEEE Engineering in medicine and biology magazine, Volume 22, Issue 4, 2003

[41] McDermid J.A., Nicholson, M., Pumfrey, D.J., Fenelon, P., *Experience with the application of HAZOP to computer-based systems*, COMPASS '95 Proceedings of the Tenth Annual Conference on Computer Assurance, 1995 s. 37–48

[42] Eisner L., Brown R., Modi D., *Risk Management Implications of IEC 60601-1, 3rd Ed.*, Compliance engineering magazine, Issue 1, 2005, saatavilla WWW-muodossa <URL: http://www.ce-mag.com/archive/05/01/004.html>, viitattu 12.7.2010

[43] Schmuland C., *Value Added Medical Device Risk Management*, IEEE Transactions on device and materials reliability, Volume 5, Issue 3, 2005, s. 488– 493,

[44] Firesmith D.G., *Use case modeling guidelines*, Technology of Object-Oriented Languages and Systems, Santa Barbara, 1999, s. 184-193

[45] Leveson N., Harvey P., *Analyzing software safety*, IEEE transactions on software engineering, Volume 9, Issue 5, 1983, s. 569–579

[46] Towhidnejad M., Wallace D., Gallo D., *Validation of Object Oriented Software Design With Fault Tree Analysis*, Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop, 2003

[47] McDermid J., Pumfrey D., *A development of hazard analysis to aid software design*, IEEE 9th Annual Conference on Computer Assurance, Gaithersburg, 1994, s. 17–25

[48] Reifer D., *Software failure modes and effect analysis*, IEEE transactions on reliability, Volume 28, Issue 3, 1979, s. 247–249

[49] Zhao N., Zhao T., Tian J., *Reliability Centered Preliminary Hazard Analysis*, Reliability and Maintainability Symposium, Fort Worth, Texas, 2009, s. 164–169

[50] Ei-Haik B., Mekki K., *Medical Device Design for Six Sigma: A Road Map for Safety and Efectiveness*, John Wiley and Sons, New Jersey, 2008

[51] Abrahamsson P., Salo O., Ronkainen J., Warsta J., *Agile software development methods — Review and analysis*, VTT Publications, Espoo, 2002

[52] Kniber H., *The Unofficial Scrum checklist*, 2009, saatavilla WWW-muodossa <URL: `http://www.crisp.se/scrum/checklist`>, viitattu 19.7.2010

[53] Kniber H., *Scrum and XP from the Trenches. How We Do Scrum*, C4Media, 2007, saatavilla WWW-muodossa <URL: `http://www.infoq.com/minibooks/scrum-xp-from-the-trenches`>, viitattu 19.7.2010

[54] Benington H., *Production of Large Computer Programs*, Annals of the History of Computing, Volume 5, Issue 4, 1983, s. 350–361

[55] DeMarco T., *Software engineering: An Idea Whose Time Has Come and Gone?*, IEEE Software, Volume 26, Issue 4, 2009, s. 95–96

[56] Winston R.W., *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, 1970, s. 1–9

[57] Watts H.S., Marc K.I., *Software Process Modeling: Principles of Entity Process Models*, Communications of the ACM, 1989, s. 331–342,

[58] Sommerville I., *Software Engineering*, 5th ed, Addison-Wesley, Massachusetts, 2000

[59] Boehm B., *Get ready For the Agile Methods, With Care*, IEEE Computer, Volume 35, Issue 1, 2002, s. 64–69

[60] Stepanek G., *Software projects secrets — Why software projects fail*, Apress, New York, 2005

[61] The Standish Group, *Chaos report*, 1995, saatavilla WWW-muodossa <URL: www.projectsmart.co.uk/docs/chaos-report.pdf>, viitattu 3.8.2010

[62] Yeo K.T., *Critical failure factors in information system projects*, International Journal of Project Management, Issue 20, 2002, s. 241–246

[63] Cheng B.H.C., Atlee J.M., *Research Directions in Requirements Engineering*, Future of Software Engineering FOSE '07, IEEE Computer society, Los Alamitos, 2007, s. 285–303

[64] Larman C., Basili V., *Iterative and Incremental Development: A Brief History*, IEEE Computer society, Los Alamitos, 2003, s. 47–56

[65] Boehm B., *Verifying and Validating Software Requirements and Design Specifications*, Software, IEEE, Volume 1, Issue 1, 1984 s. 75–88

[66] Pressman R.S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Singapore, 2005

[67] Boehm B., Hansen W., *Spiral Development: Experience, Principles and Refinements*, ACM, Pittsburgh, 2000, s. 1–37

[68] Boehm B. et al., *Using the WinWin spiral model: a case study*, IEEE Computer, Volume 31 , Issue 7, 1998, s. 33–44

[69] Thompson A., *Entrepreneurship and business innovation*, Vineyard Publishing, Guildford, 2005

[70] Samejima M., Shimizu Y., Akiyoshi M., Komoda N., *SWOT Analysis Support Tool for Verification of Business Strategy*, IEEE International Conference on Computational Cybernetics, 2006, Budapest, s. 1–4

[71] Mannion M., Keepence B., *SMART requirements*, ACM SIGSOFT Software Engineering Notes, Volume 20, Issue 2, 1995, s. 42–47

[72] Cohn M., *User Stories Applied*, Addison Wesley, Massachusetts, 2004

[73] Lehman M., *Laws of Software Evolution Revisited*, Lecture Notes In Computer Science, Volume 1149, 1996, Springer-Verlag, London, s. 108–124

[74] Bennett K., Rajlich V., *Software Maintenance and Evolution: A Roadmap*, Proceedings of the Conference on The Future of Software Engineering Limerick (2000), ACM, Limerick, s. 73–87

[75] Paetsch F., Eberlein A., Maurer F., *Requirements engineering and agile software development*, Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE (2003), Confrence publishing services, s. 308–313

[76] Ladas C., *Scrumban — Essays on Kanban Systems for Lean Software Development*, Modus Cooperandi Press, Washington, 2008

[77] IEEE, *IEEE Recommended Practice for Software Requirements Specications*, IEEE Std 830-1998, Software Engineering Standards Committee, New York, 1998

[78] IEEE, *IEEE Standard for Developing a Software Project Life Cycle Process*, IEEE Std 1074-2006, Software Engineering Standards Committee, New York, 2006

[79] Mann C., Maurer, F., *A case study on the impact of scrum on overtime and customer satisfaction*, Agile 2005 Conference (2005), IEEE Computer Society, Los Alamitos, s. 70–79

[80] Eberlein A., Prado Leite J.C.S., *Agile Requirements Definition: A View from Requirements Engineering*, International Workshop on Time-Constrained Requirements Engineering, Essen, 2002

[81] Smiths H., *5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up*, Rally Software Development Corp, 2006, saatavilla WWW-muodossa <URL: `http://www.agilejournal.com/whitepapers/661-5-levels-of-agile-planning`>, viitattu 28.7.2010

[82] IEC, *Test Report: IEC 60601-1-4 Medical electric equipment, Part 1-4 General requirements for safety*, 2002, saatavilla WWW-muodossa <URL: `http://www.devicecompliance.com/CB60601-1-4.doc`>, viitattu 31.7.2010

[83] Feldmann R., Shull R., Denger C., Höst M., Lindholm C., *A Survey of Software Engineering Techniques in Medical Device Development*, High Confidence Medical

Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, Cambridge, 2007, s. 46–54,

[84] Lin W., Fan X., *Software Development Practice for FDA-Compliant Medical Devices*, International Joint Conference on Computational Sciences and Optimization, Volume 2, 2009, IEEE Computer Society, Los Alamitos, s. 388–390

[85] Leveson N., *Safeware: System safety and computers*, Addison-Wesley, Massachusetts, 1995